

Intelligent Audio for Games

*A proposal for using AI to manage interactive audio in
computer games*

C. Walder

MA Music Technology

Supervisors: Dr. T. Myatt, Dr. A. Field

August 2005

Welcome to the fascinating world of digital audio for game development!
Graphics may bring an interactive digital world to life, but sound and music give a game its soul. A world without audio would seem hollow and empty indeed. Hyperbole aside, there are very few ways other than audio to so dramatically influence a players emotions.

[Boer, 2001: 513]

Abstract

An investigation is presented which proposes the development of artificially intelligent agents as a way to improve interactive music and sound design in the Computer Game genre.

An analysis is given of current and near-future sound design technology and methodology within the games industry including discussion of Microsoft's DirectMusic Producer, 3D audio, interactive sequencing and algorithmic composition. It is found that, although advances have been made to improve audio for games, the challenge of creating music and sound design that can be performed interactively within a computer game places severe limits on composers and sound designers.

Comparisons are made with the Film genre which is proposed as a possible role model for interactive sound design. Techniques such as asynchronicity, structure, transitions and association are discussed and proposed as key techniques used in film sound but which are lacking in game audio.

Artificial Intelligence (AI) programming is proposed as a possible solution for the problems associated with interactive sound design. A number of AI techniques are discussed and the Soar architecture is proposed as a good platform for the development of game audio AI. An introduction to the design, functionality and programming of Soar is given and features such as production rules, state based mechanics and semantic net style knowledge representation are identified as being highly beneficial to audio agent design. Previous and current research using Soar for game AI development is reviewed and a discussion of Soar Quakebot and Interactive Drama Architecture presented with regards to how their designs could be used to develop an audio agent featuring human behaviour modelling, anticipation and state based plot structure.

CONTENTS

Introduction	1
Chapter 1: Sound and Music in Computer Games	4
1.1 Game Audio: A new age	4
1.2 Game audio in the past	5
1.3 Game audio in the present	8
1.3.1 3D Audio	8
1.3.2 Interactive music sequencing	12
1.3.3 Algorithmic composition	16
1.3.4 Tools for interactive audio	18
1.3.5 Renderware Audio	18
1.3.6 Miles Sound System	19
1.3.7 DirectMusic	19
1.4 DirectMusic Producer	20
1.4.1 DirectMusic Components	21
1.4.2 Segments	21
1.4.3 Styles	22
1.4.4 Patterns and Motifs	23
1.4.5 Bands	25
1.4.6 Waves and DLS	26
1.4.7 Audiopaths	27
1.4.8 Transitions and Markers	28
1.5 Game Audio in the Future	31
Chapter 2: Film Audio	33
2.1 The convergence of Film and Games	33
2.1.1 Sound with picture	35
2.1.2 Telling a story	36
2.1.3 Audience	37
2.1.4 Interactivity	38
2.1.5 Length	38

2.1.6	Comparison: Film and Game	38
2.2	Film audio techniques	39
2.2.1	Synchronous sound	41
2.2.2	Asynchronous sound	43
2.2.3	Silence	45
2.2.4	Association	46
2.2.5	Transition	47
2.2.6	Structure	49
2.3	Implementing film audio techniques in games	52
 Chapter 3 Using AI for Interactive Audio		 54
3.1	What is AI?	54
3.1.1	Inference machine	55
3.1.2	Interface	56
3.1.3	Knowledge base	56
3.1.4	Requirements of an agent	57
3.2	AI for Game Audio	57
3.2.1	Tasks for Audio AI	58
3.3	AI Techniques in Games	59
3.3.1	Expert Systems	60
3.3.2	Case-based reason	60
3.3.3	Finite-state systems	61
3.3.4	Production systems	61
3.3.5	First order (predicate) logic	62
3.3.6	Semantic networks	62
3.3.7	Genetic Algorithms	63
3.3.8	Neural networks	64
3.3.9	Suitable techniques	65
3.4	Soar architecture	65
3.4.1	Basic Soar operation	66
3.4.2	Soar functionality	68
3.4.3	WME support	71
3.4.4	Preferences	72

3.4.5	Impasses	73
3.4.6	Syntax	75
3.4.7	Development software	80
3.5	Soar research on computer games	82
3.5.1	Soar Quakebot	82
3.5.2	Interactive Drama Architecture	86
3.6	Programming Soar	89
3.6.1	Water Jugs	90
3.6.2	Eaters	93
3.6.3	SML (Soar Markup Language)	96

Chapter 4: Conclusion	103
------------------------------	-----

References

Appendices

- Appendix A: Correspondence with Scott Morton
- Appendix B: Production code for 'The Water Jug' problem
- Appendix C: Advanced 'cool-eater' productions
- Appendix D: Production code used in SML test program

Audio Visual Examples:

- Audio/Visual Example 1: *Final Fantasy VII* Introductory Scene
- Audio/Visual Example 2: Orchestrated version of the Overworld theme from *Final Fantasy VII* composed by Nobuo Uematsu.
- Audio/Visual Example 3: *No One Lives Forever* transitions
- Audio/Visual Example 4: Introductory scene to *Matrix Online*
- Audio/Visual Example 5: *Gran Turismo 3 A-Spec*
- Audio/Visual Example 6: Scene from *Star Wars Episode II: Attack of the clones*
- Audio/Visual Example 7: Battle scene from *Gladiator*
- Audio/Visual Example 8: Patrick Bateman confesses in *American Psycho*

Figures

Figure 1.1	DirectMusic Producer Segment interface	22
Figure 1.2	DirectMusic Producer Style interface	23
Figure 1.3	DirectMusic Producer Pattern interface	24
Figure 1.4	DirectMusic Producer Bands interface	25
Figure 1.5	DirectMusic Producer Wave interface	26
Figure 1.6	DirectMusic Producer DLS Designer window	27
Figure 1.7	DirectMusic Producer Audiopath interface	28
Figure 1.8	Adding embellishments to a pattern in DirectMusic Producer	29
Figure 1.9	DirectMusic Producer Chord properties	30
Figure 1.10	DirectMusic Producer Marker track	30
Figure 3.1	Semantic network	63
Figure 3.2	Soar operation cycle	71
Figure 3.3	Soar sub-goaling	75
Figure 3.4	Visual Soar programming environment	80
Figure 3.5	Soar Debugger interface	81
Figure 3.6	Sample operator hierarchy for Quakebot style agent	84
Figure 3.7	Example sequence of plot states	88
Figure 3.8	'Eaters' game environment	94
Figure 3.9	Flow diagram of SML test program	98

Introduction

This report discusses the potential of audio within interactive products, focussing on music and sound design for computer games. In order to propose future methods and technology for computer game sound design, the current state of the industry is discussed as well as the possible cross-over of knowledge and skills from the Film industry. In the past the film industry has often been sceptical of the value of sound in visual media, however contemporary film makers often regard sound as being equally important (Weiss & Belton (eds), 1985). We find that the perceived importance of audio within computer games is increasing as is the capability of technology to provide high quality audio to the end user,

As our budgets grow each year you will continue to see a huge rise in big production and high quality audio. The technology is getting to a point where the platforms are no longer limiting the possibilities of what we can do.
[Tallarico, 2005]

Comparisons between the technical specifications of the PS2 (currently the most popular games console) and its successor the PS3 illustrate the increasing focus on audio by hardware designers, including the capability to output 5.1 audio and compatibility with Dolby and DTS.

Interactivity is seen as being the most significant challenge facing game audio sound design and composition (Wilde, 2004), particularly for composers who often come from backgrounds in linear audio¹ such as the Film and Music industries. A simple solution, often seen employed in games (e.g. *Deus Ex* (Stevens, 2001)), is to write sections of linear music which is chopped up and triggered to coincide with events on screen. While this approach can be effective, the composer usually has little or no input in the actual implementation of their music within the game, resulting in very basic interactive music.

¹ Within this document, 'linear music' shall refer to music that is not interactively performed.

Although interactive music and sound design is often seen as a problem, it is also a great opportunity to involve the audience (in the case of games a player) to a much greater degree than is ever possible with linear music. In order to fully realise the potential of interactive audio in games, it is necessary to consider how the player interacts with the game and music and to look at ways of driving the audio to interact with the player. One such method is the use of artificial intelligence (Ames).

Computer games have for some time made use of AI to control non-player characters (NPC) such as opponents and allies and in return the artificial intelligence community has used games as a platform for developing state of the art AI techniques (Laird & Lent, 1999). In modern computer game design, AI is an integral component of the development process,

It is frankly amazing to this developer how the field of game AI has grown and become an integral part of the game-development cycle over the past few years. Recent polls have shown an enormous surge in the amount of time, development, and just plain CPU cycles devoted to the creation of good game AI. [Treglia (ed.), 2002: 229]

Given that there is a wealth of experience, within both the games industry and AI research community, in using AI as an interactive component within games, it seems reasonable to propose using AI for interactive audio.

Different AI techniques are discussed in terms of their current application within games and possible implementation for audio. A detailed investigation of the Soar AI engine (Laird & Congdon, 2005), developed at the University of Michigan Artificial Intelligence laboratory, is reported and examples of how this system could be used to drive interactive audio.

In chapter 1 we look at the current state of sound and music in computer games; its purpose and how that purpose is fulfilled technologically and artistically. We consider the

features offered by audio tools for computer game designers such as Criterion's Renderware (www.csl.com) and focus on the operation of Microsoft's DirectMusic Producer (www.microsoft.com). Having established the current state of audio we consider the future of game audio and the direction in which the industry is moving.

Chapter 2 considers the relevance of film music and sound design to game audio. Comparisons are made between the two genres and we find that there are many lessons learnt in the history of film sound which are applicable including both aesthetic theory and practical techniques. Examples are given of convergence between the two industries and difficulties with implementing film audio techniques interactively are identified and a possible solution suggested.

Chapter 3 presents the case for using AI in game audio. Different AI techniques are reviewed along with their traditional implementation within games. Production systems, finite-state machines and semantic networks are found to be the most appropriate for audio applications and the Soar engine developed at UMICH is proposed as a strong candidate for audio AI. A detailed description of Soar's operation is presented with examples of how it might be used to control interactive audio and interface with computer game programs.

Chapter 4 offers a conclusion discussing the main findings of the report. Further research into developing AI for audio capable of planning and learning is suggested and the major steps for implementing the AI described in this report are identified. Neural networks systems are identified as an alternative path for future research.

Throughout this report, audio and audio-visual examples are presented. These may be accessed in the *AV Examples* folder accompanying CD-ROM and can be played with most pc media players such as Winamp (included in the *Resources* on the CD-ROM). Soar suite 6.1 is also included in the CD-ROM resources.

Chapter 1

Sound and Music in Computer Games

1.1 Game Audio: A new age

In recent years audio has enjoyed a large boost in status within the Games industry resulting in larger budgets for audio development and more prestige associated with having good game audio. Aaron Marks describes the popularity of the audio sessions at the 2003 Game Developers Conference:

Game audio, it seems, continues to gain attention -- and what was once a relatively non-existent GDC topic a few years ago now plays out to large, standing-room-only crowds in a very conspicuous way. From the outgrown conference rooms to the many entertaining side shows, game audio is gathering steam as a force in the video game industry.
[Marks, 2003]

With the upcoming release of the next-generation consoles, game audio is arriving at a significant turning point. Traditionally audio has been largely marginalized within the games industry in terms of technology which has had a significantly detrimental effect on the ability of games designers to successfully implement interactive soundtracks. An excellent example of this is the PlayStation 2 (PS2) which is currently the most popular console on the market yet only provides 2MB of RAM storage for audio files. This is approximately equivalent to only 2 minutes of CD quality music (i.e. 16bit, 44.1KHz) compressed using an MP3 algorithm by a factor of 11:1 (a popular consumer compression ratio). Although it is possible to stream audio directly from the game disk, this severely restricts interactive sound design as no processing or manipulation of the audio data can be performed. Conversely, the eagerly anticipated successor to the PS2,

the PlayStation3 (PS3), has 512MB of RAM and a removable hard disk drive which could be used to store audio samples, in addition to providing the capacity for presenting audio in 5.1 surround sound and supporting real-time manipulation of Dolby and Dts audio (the two main audio encoding schemes used in the Film industry). In a press conference at E3 2003, Sony showed a demo of hundreds of thousands of leaves in a vortex and boasted that the PS3 could process the audio with a separate channel for each leaf. Clearly the technology available for audio is vastly superior to previous consoles; however this means that the challenge is now to use that technology to its full potential and make significant improvements to the presentation of interactive audio.

1.2 Game audio in the past

It is important to note that although audio technology has been lacking this has not meant that the sound design and composition has been poor, only that the technology has restricted the potential of the audio. For a long time, game designers have known the value of good music and sound design in bringing their artificial worlds to life:

Consider the ominous ambient sounds of survival horror titles like Resident Evil, which compound the tension as you happen upon those relentless zombies chewing up your Alpha Team comrades... Even early games like Space Invaders earned much of their addictive appeal by getting into your head with thumping, repetitive sound schemes.... Properly designed, sound and visual cues work together to produce an experience greater than the sum of their parts.
[McDonald, 2004:1]

Audio/Visual Example 1 demonstrates an effective use of sound design in the introductory scene to *Final Fantasy VII* on the PlayStation 1. Despite the technical restrictions of the hardware (500kB of audio memory and 24 audio channels), a good sound design has been implemented. At the beginning of the scene the music is calm

and serene as a female character (Aeris) is introduced looking at some mystical green lights. As she turns and the camera pans out, we see that she is in fact in the middle of a large and highly industrialised city; traffic noise is brought in to contrast with the peaceful music, giving the impression that Aeris is somehow separate from the city's industry and establishing an important point of her character. Suspended notes on a string sound build up anticipation and a fanfare strikes up as camera pans out to see the city in full, reinforcing the sense that this is a really big, important city. In the background we hear a train, which at first seems to be a part of the general traffic but is in fact heralding the off-screen arrival of a train carrying the main character (Cloud). Flashes of the train are shown accompanied by a harsh grating sound which immediately alerts us that something is wrong and out of place compared to the general hubbub of the city. The music then picks up pace indicating that something exciting and dangerous is about to happen as the train grinds into a station and Cloud jumps off the train and begins an attack on a power plant.

Throughout the scene, sound is used to expand upon the visuals, including using music to influence the player's emotions (e.g. the climactic build-up as the camera builds up to inspire a feeling of awe) and introducing new events with off-screen sound effects. The contrasts of sound at the beginning of the scene give us a deeper insight into the character of Aeris than would be possible with any visual.

In Asia the quality of computer game composition has been recognised for a long time and a number of audio CD albums have been released featuring orchestras, rock bands and/or pop musicians performing music from computer games, such as *Splinter Cell: Chaos Theory* by Amon Tobin. Audio/Visual Example 2 is an orchestrated version of the Overworld theme from *Final Fantasy VII* composed by Nobuo Uematsu. Audiences in the west have also begun to acknowledge the work of computer games composers. The recent Dear Friends concert in Los Angeles featuring a selection of Nobuo Uematsu's

music played to a sold out audience [Schneider, 2004] and the Video Games Live tour featuring music from a number of computer game composers has also been very well received [Goldwasser, 2005]. Despite the restrictions placed upon video game audio in the past, or perhaps because of them, composers have been able to produce good music, however in terms of interactivity, sound tracks have been lacking. Historically, a linear method has been taken when implementing computer game soundtracks. It has been quite common for there to be a certain piece of music for a certain level in the game (Stevens 2001), with the music changing between levels. In computer games which have less linear game play and allow the character more freedom of exploration, music is more likely to be dependant on the area the character is in; for example there might be eerie music in a dark cave and upbeat music in a sunny meadow.

The problem with this is that it assumes the player's emotions are always going to be the same in a particular place. As the plot progresses within the game, the music attached to certain areas may change, but this only serves to make the player very aware of the linear plot. Complex and well developed sound design and musical ideas can be used in this system, but only during linear scenes such as in Audio/Visual Example 1 where there is negligible interactivity. Ideally we would like a system that altered the music and sound design according to the player's emotions and actions rather than their position in the game (although their surroundings and events in the plot would have to be taken into account). This concept is key to realizing the potential of interactive game audio, particularly in genres such as role-playing and action/adventure where the plot is central to the experience.

An example of the area/level sound design system is Bioware's *Neverwinter Nights*. Based on the popular pen and paper role playing game Dungeons and Dragons, *Neverwinter Nights* offers the player a wide variety of character types from Lawful Good Sorcerers to Chaotic Evil Barbarians, however the music is exactly the same whether

you are playing a hero nobly battling the forces of evil or a dark warrior slaughtering innocents.

Another RPG, *Morrowind*, allows the player great freedom in exploring the game world. In *Morrowind* the player may choose to follow the story line or not and there are a large number of sub-plots for the player to become involved in, however the game music has been criticized for having been written to effectively bypass interactivity by using a generic soundtrack which would fit a variety of non-specific situations:

One of my favorite role-playing games is Elder Scrolls III: Morrowind by Bethesda Softworks. I love the music from this game, but in terms of its emotional function throughout a gameplay session, it honestly doesn't accomplish much. The music is ambient in nature and simply plays straight through for roughly thirty minutes, then starts over again. There are cues for battle encounters, but other than that, nothing. When I enter a dark dungeon that is critical to the game's storyline, the music doesn't foreshadow anything. When I have just finished a battle and am inches from death, struggling to get back to an area where I can rest, the music doesn't reflect the critical nature of the situation at all. There is not even a shift in the wrong direction - there's no shift at all.
[Morton, 2005a]

1.3 Game audio in the present

Currently there are a number of tools available to the computer game composer and sound designer to aid in the production of interactive including Criterion's Renderware Audio, RAD's Miles Sound System (www.radgametools.com) and Microsoft's DirectMusic Producer. Many techniques have been developed to enhance interactivity by generating or manipulating audio in real time such as 3D audio processing, interactive music sequencing and algorithmic composition.

1.3.1 3D Audio

Three dimensional (3D) audio was introduced to the mainstream gaming community with the launch of the first Playstation console platform in 1994 and in the years since, 3D

graphics have become ubiquitous throughout the industry. Almost all games feature 3D graphics even if the gameplay is not truly three dimensional. As a result of the popularisation of 3D in computer game graphics, 3D audio engines were developed to provide a way to use the same 3D world representation to control both audio and video aspects (Treglia (ed.), 2002) resulting in a more cohesive, realistic simulation of the game world.

Key to the design of 3D audio systems is the concept of having an environment with a single listener and multiple sound sources. The position of the listener is that of the player in the game world, which may be the same as that of the character being controlled in a first-person perspective game or some other point (typically behind and slightly above the character) in a third-person perspective game. The 3D audio engine interprets data about the position of objects acting as sound sources and uses a number of processes to place the sources in a sound-field relative to the listener. Typically the engine will be able to decode the sound field to a number of speaker arrangements although in practice this only involves two dimensional arrangements such as 5.1 and stereo and true three dimensional sound designs (i.e. positioning sounds on the z-axis as well as the x and y axes) are never implemented. Sounds are panned according to their position relative to the player's perspective either through 360° around the player in surround sound speaker setups or between the left and right speakers for stereo.

Psychoacoustic processing such as Head Related Transfer Functions (HRTF), which attempt to mimic the effect our heads have on sound coming from different directions (Menshikov, 2003), are often used to try and emulate surround sound on stereo systems. This is particularly useful if there is an important audio event occurring behind the player point of view; for example an NPC is attempting to talk to the player or an enemy is attacking. By themselves psychoacoustic effects can prove unreliable as it is difficult to compensate for the fact that humans use small head movements to localize

sounds (Rossing, 2002) particularly in distinguishing between those coming from in front or behind. However, when reinforced by visual data, the brain will more readily accept psychoacoustic cues – if there is ambiguity as to whether a sound is coming from in front or behind and the visuals indicate that it is not in front, the brain will most likely be convinced that the sound is coming from the rear.

Several techniques are used to simulate distance between the player and the sound source. The first and simplest technique imitates the property of three dimensional waves to vary inversely with distance from their source (Tipler, 1999) which is implemented by reducing the volume of the source as the distance between the sound source and listener is increase. This effect is often enhanced by using a filter to remove or suppress the higher frequencies for larger distances between listening position and source which simulates the effect of higher frequencies being absorbed by the air (White, 2003). The final technique is to add reverb to distant sources. Reverb simulates the effect of sound waves reflecting off nearby objects and is the main auditory cue used to calculate distances. When a sound source is close to a listener, the sound coming directly from the source to the listener is much louder than the sound bouncing off nearby objects because it has travelled a much shorter distance, however as the separation increases, the level of the reverberant sound compared to the direct sound will increase. The sound of reverb is very characteristic of the surroundings and using appropriate reverb parameters is important in creating a believable sound environment.

To create dynamic environments with moving sound sources, Doppler shift is taken into account.

When a source and a receiver are moving relative to each other, the frequency observed is not the same as that emitted. When they are moving toward each other, the frequency is greater than the source frequency; when they are moving away from each other, the observed frequency is less than the source

frequency...A familiar example is the change in pitch of a car horn as the car approaches or recedes.

[Tipler, 1999: 463]

Real-time pitch shifting, often implemented using sample skip methods, is used to simulate the Doppler Effect for moving sources in 3D engines. Further, advanced techniques in 3D sound placement include calculating the effects of obstruction, occlusion and exclusion caused by objects blocking the direct path between the listener and sound source. When there is a finite obstruction blocking the path of a sound, part of the sound will diffract around the obstruction (and possibly also transmit through it too) and still reach the listener. Low frequencies diffract more readily than high frequencies so a low-pass filter is applied to the sound source, simulating the effect of the diffraction, with the cut-off frequency falling as a greater diffraction would be required to bypass the object (Treglia (ed), 2002). If there is an infinite obstruction (i.e. there is no clear path for the sound to reach the listener), an effect known as occlusion occurs. Even without an unobstructed path, some sound may reach the listener as sound waves can transmit through solid objects. As with diffraction, low frequencies transmit more readily than high frequencies, so occlusion may be imitated using a low-pass filter across both direct sound and reverb (Menshikov, 2003). Exclusion occurs when there is the sound source is in a different room but has a direct path to the listener (e.g. through an open doorway) and is simulated by processing the reverberant sound from the source through a low-pass filter. Creating a believable sound environment which is coherent with the 3D graphical environment on-screen can help considerably to immerse the gamer in the artificial world and the techniques excel at unifying the sound and picture. However, one of the shortfalls with the implementation of this solution is that the audio is artistically compromised since the decision of which sound sources are relevant is, at worst, left to a standard routine within the API (application program interface) as with DirectMusic, or,

at best, is the responsibility of a tools programmer and is unlikely to involve creative sound design concepts or even some useful gaming effects.

Some players have said that for games with only 3D impact sounds, such as Unreal Tournament 2004, it can be difficult to tell if a shot hit the opponent.
[Song, 2005]

By not involving the sound designer and composer in implementing audio within the game, the programmers are effectively performing compositional roles. In interactive audio where there is no pre-determined structure, the choices involved in selecting sound and music for different events is of great musical and artistic significance and should be performed (or at the very least involve) the composer or sound designer.

1.3.2 Interactive music sequencing

Making interactive music can be viewed as controlling a puppet. We need to pull the right strings to make it seem alive. And if we pull the right strings to control the way the music plays, we may even pull the emotional strings of our game player.
[Deloura (ed.), 2001: 551]

Music can be a significant influence on mood and emotion and is therefore one of the most powerful tools available when designing an interactive product. This very quality, however also makes music a double edged sword; just as music can enhance a game, if poorly designed it could remove any sense of immersion or even irritate the gamer. The main reason that this might occur in games is because of badly implemented interactivity. A well known problem that has plagued computer games for many years is that of overly repetitive music (Morton, 2005a). The issue arises because, unlike other media such as music albums and films, it is not uncommon for an individual to spend extended periods of time playing a single game; a recently released game, GTA: San Andreas is reported to take at least 150 hours to complete (Leyton, 2004). If a gamer is

exposed to repetitive music during a game which they are playing, possibly for hours at a time, it is likely that this will affect their mood and they will start to feel that the gameplay is very repetitive and get bored and/or irritated with the game.

There are so many reasons why this is a bad approach. Looping tends to go hand-in-hand with the "watered-down, ambient music" approach; and, it makes it worse. Not only have you eliminated the emotional effectiveness of the music by generalizing it and not applying it to a context, but by looping it over and over, you've completely detached the player from even registering it altogether. And what's worse, it usually becomes annoying after a time. Now we've moved down from "why should we even have music playing here" to "why shouldn't we turn off the music altogether and listen to MP3s?"
[Morton, 2005a]

To avoid the problem of overly repeated patterns and 'level' music, interactive music sequencing is used to perform composition in real time. The design of good interactive sequencers takes into account emotion, meaning, variety and transitions. As mentioned above, influencing the player's emotions is one of the prime functions of music in computer games whether it is to raise the tension and feeling of danger during an important battle or evoke feelings of sorrow and loss at a friendly character's death. Interactive sequencers aim to choose the correct piece of music to complement the emotional involvement of the player at certain points in the game. This is implemented using pre written scripts which send triggers to the sequencer that a certain even or point in the game has been reached. When done well this can be effective although it is still fairly linear overall as the player has to move through a pre-written sequence of events. In a similar manner, music (and sound effects) can be used to communicate meanings to the player. The motif of a heart-beat is often used to signify that the character's health is running low or a fanfare used to celebrate a victory. Like the emotion aspects of the interactive sequencer, music with meaning is usually scripted to coincide with certain events; although this method is not continually repetitive, it is repetitive on the larger

scale and the gamer will soon realise that this piece of music will play when he defeats an enemy etc., reducing the effectiveness of the cue. In order to provide variety, interactive sequencers (Such as DirectMusic Producer – see section 1.4) often incorporate algorithmic composition functions allowing the composer to write a number of slightly different parts for each segment which will be played back at random in the game, reducing the problem of continuous repetition. In order for the sequencer to offer a lot of variety, however, it is necessary to have a large amount of material written by the composer which simply isn't possible if they are brought in to write music at the end of the project under short deadlines as is generally the case.

Another particularly important component of an interactive sequencer is its transitions.

Transitions can be defined as one or more changes occurring over a time interval. A transition might mean an interpolation of some type of state data done over a specific time interval. In addition, a transition might mean a new track section, musical cadence, key change or other compositional technique. A transition could be a combination of these things. Transitions may be governed by a combination of game logic and music language logic... These types of implicit and explicit controls over transitions are another key element of the control of interactive music. [Deloura (ed.), 2001: 553]

Transitions are important in interactive music because the player has control over when events happen and often also in what order. In linear music transitions are used to develop the music throughout the piece and are often highly structured however in interactive computer game music the music may need to change at any point to another theme (Stevens, 2001). A simple – and common – interactive method for controlling transitions is to stop one track and start another or fade between the two. This is usually musically unappealing and breaks the continuity which is counter productive to creating a feeling of immersion.

Moving smoothly from one musical cue to another in a nonlinear game environment is tricky, but necessary to maintain the thread of gameplay.

Transitions are the glue that add continuity to a game score, and hence to the game itself.

[Whitmore, 2003]

Another method uses pre-written transition segments designed to move between tracks. A sequencer employing this method usually has the capacity for the composer to select certain points in the track as being acceptable to switch out/switch in. Simple implementations of this usually sound quite clumsy since the pre-written transitions have to be very generic and do not take into account the full musical context of either track; although when carefully designed, this is much better than simply switching tracks in and out. More complicated versions of this method are possible by moving to an acceptable switch-in point in the middle of the new track making it possible to better preserve continuity.

Audio Visual Example 3 is from the game *No One Lives Forever* and moves through six transitions (at times 1.22, 2.14, 3.10, 4.08 and 5.33) smoothly using well written switch in and out points. The final point of good design of an interactive music sequencer is that of balance.

If my dynamic music engine simply follows gameplay and triggers "appropriate" music based on what the player is doing or experiencing, then that music loses its ability to speak independently. It can no longer impose unique kinds of emotions by relating the music in contrasting or half-parallel ways to the situations themselves... If ambient music is on one end of the spectrum, then dependent "adaptive" music is on the other. Neither extreme serves very well in drama-based games

[Morton, 2005a]

With the interactive sequencer it is important to have a balance which does not favour either extremely flexible ambient music or purely reactive music. Traditional linear music features both development and structure at its core; music which favours one or the other to strongly will rarely receive approval from the mainstream audience. Without a

balance between development and structure, reactivity and flexibility, an interactive sequencer is unlikely to produce music which will be considered to be 'as good as' linear music. A game where the sound designers have made particular efforts to get this balance is Halo 2. A concept called quantum music is used which uses loops of music triggered by events in the game but will fade out the music if the player does not continue to perform actions which serve to advance the game:

"I want people to have a great experience however they play the game," O'Donnell concluded. "My philosophy is... the ear does not blink." Any interruption, any abrupt change, any conscious notion that the player is in control of the music, will bring the player out of the experience.
[Waugh, 2005]

1.3.3 Algorithmic composition

An algorithm is a series of instructions and/or rules which describe an action or activity allowing for automation of that activity. In terms of music, algorithms can be used to automate composition,

...algorithms for composing music have the most serious philosophical implications since they tend to replace what may feel are activities more rightfully the domain of humans: principally the freedom of choice but including more illusory concepts, such as imagination, soul, inspiration and intuition.
[Cope, 2000: 2]

This obviously has many benefits for interactive computer game music and algorithmic methods have been included to provide musical variety as early as 1986 in Ballblazer by Lucasfilm Games which used a jazz style improvisation algorithm for the game music (Langston, 2005). Contemporary games still use algorithmic methods, but in a slightly different way.

Algorithmic music can be generalized into two general styles; stochastic (or aleatoric) music and music which is permutation of pre-written elements (Dodge & Jerse, 1997).

Aleotonic compositions use random procedures and probability theory to determine musical events, applying constraints such as limiting the possible choices of notes or making certain choices more or less likely. By imposing appropriate constraints on pitch, rhythm and structure, it is possible to create algorithms which produce a very individual musical result despite the fact that that is rarely ever playing the same thing. This form has a lot of potential for use within computer games but because the design of aleotonic algorithms is so far removed from traditional composition methods that it seems computer games composers are reluctant to adopt it. Aleotonic music is a powerful tool, however, as it can explicitly make use of musical rules which are implicit in traditional composition methods. Academic research (Cope, 2000) has noted that Classical composers such as Mozart and Bach referenced the musical rules established by Fux and have also used recombination algorithms to create music which is stylistically indistinguishable from Bach's chorals.

A more common approach to algorithmic composition in computer games is to use permutations of pre-written sections of music. It is possible to use probabilistic methods and constraints to provide structure to the arrangement of sections however most implementations in computer games, such as the segment system found in DirectMusic Producer, use fairly simple random procedures aimed at reducing repetition rather than creatively influencing the structure. The benefit of using this form of algorithmic composition is that the composer has direct control over what music will actually appear in the game although a high level of musical variety is dependant on the composer writing a large number of sections for each theme and it can be challenging for the composer to make all the themes completely interchangeable (Morton, 2005b).

1.3.4 Tools for interactive audio

When designing computer games, audio programmers make use of a number of software tools to implement 3D audio, interactive sequencing, algorithmic composition and to interface with the audio hardware. Some games developers design their own proprietary tools, but many take advantage of middleware tools packages such as Renderware Audio, Miles SoundSystem and DirectMusic. Renderware Audio and Miles Sound System are commercially distributed products aimed solely at providing game developers with programming tools while DirectMusic is freely distributed and includes DirectMusic Producer; a composition tool aimed at providing games composers with a framework for making interactive music.

1.3.5 Renderware Audio

Renderware Audio is part of Criterion's world middle ware solution, the Renderware Platform, which also includes tools for, graphics, AI, physics and game design framework. A number of features are included and are focussed on optimising the performance of audio with optimisations for all major consoles and PC's. Renderware supports multi-channel playback, Dolby surround sound and provides designers with DSP and reverb effects in addition to a powerful 3D audio engine. The most useful creative and unique tool provided by Renderware is the audio events toolkit which allows many parameters of wave and stream playback (including volume and pitch) to be changed at run time. A more detailed discussion of Renderware's features is not possible due to non-disclosure agreements which prevent details of the software's operation from being distributed.

1.3.6 Miles Sound System

Produced by RAD, Miles Sound System is popular among game developers having appeared in over 3000 games. Miles has many similar features to Renderware including multi-channel audio, Dolby support, 3D audio design API, and a variety of DSP effects. In addition, Miles is designed to work particularly well with DirectSound and has interactive support for functions such as tempo control. Like Renderware, Miles does not provide any additional tools to aid composers or sound designers. Perhaps the most appealing feature of Miles, however, is the strong after sale support offered to game developers particularly for small companies who might not have dedicated audio programmers. As with Renderware, the scope of discussion is limited to information released by the developer.

1.3.6 DirectMusic

DirectMusic is a part of Microsoft DirectX, a set of components installed in the Windows operating system to cater for multimedia and game content including providing API's and software design kits (SDK's) for graphics and audio. DirectMusic can support most of the basic features of Renderware and Miles with regards to playing back and generating audio, although often it requires more programming as the features are less well developed. In addition DirectMusic only supports the Windows platform and does not provide support for surround sound as standard. The two main advantages that DirectMusic has over either Renderware or Miles are that it is free and that it supports DirectMusic Producer.

1.4 DirectMusic Producer

Although DirectMusic can be used by itself to playback, mix and process midi and wave audio, it is by using DirectMusic Producer to author content for DirectMusic that the full creative potential of the platform is realised. DirectMusic Producer provides an interface for composing and arranging interactive music and sound design and can provide a composer with control over variations ranging from different patterns of notes to tempo which can be randomised or triggered at run-time. A parameter called groove level gives the composer a level of control over the choice of patterns available to the soundtrack at different points in the game and it is possible to synchronize different pieces of music and other events such as graphics to rhythmic boundaries. In order to allow the piece to be composed in real time, content is built as a hierarchal system of components which store separate parameters such as note pattern, chords and instrumentation. By sending patterns and audio effects to different audiopaths, it is possible to apply effects and panning to create a 3D sound field. As an extra measure of run-time control for the composer, scripts can be written to automate musical events. Unlike the other tools in the area, DirectMusic Producer addresses the compositional challenges which are associated with interactive music as well as providing a framework for the communication of musical ideas between the composer and programmers,

Fortunately a tool like DirectMusic Producer allows you to focus solely on the issues related to interactive musical composition. Rather than composing on one hand and thinking interactive on the other hand and having to somehow smack the two sides together like a pair of mud pies you can compose interactively from the first step. You only have to design the components and the structures that will control how they interact. The design tool does the grunt work... To me [DirectMusic Producer] signals the beginning of a new era in musical composition. There are a small number of us interactive composers who think of ourselves in that light but until now have been much like a drummer banging on rocks and logs. Now we have a real drum set and we are pulling out the big sticks with eager anticipation.
[Griffin, 1998]

In terms of interactive composition, DirectMusic Producer has taken some important first steps and has paved the way for further development. It is important that this development occurs, however, if a truly interactive music system for games is to be realised as the methods for providing variety in the real-time composition within DirectMusic rely on very basic random procedures and do not take full advantage of the possibilities of algorithmic composition. Using triggered scripts provides the composer with a measure of control over events at run-time but effectively is falling back on the traditional method of relying on linear techniques and, like many of the controls given over to the composer, relies on the game programmers to implement well.

1.4.1 DirectMusic Components

DirectMusic Producer is used to author the components which DirectMusic uses to perform real-time composition.

1.4.2 Segments

Segments are the top-level components, being composed of a collection of sub-components including Styles and Control tracks (such as tempo and Style change), and are used to play back the DirectMusic score. Segments may also contain sub-segments but global data such as tempo and chords is always stored in the top-level segments.

Figure 1.1 shows a screenshot of the DirectMusic Producer window with the Segment interface opened. Tracks have been inserted to illustrate all of the possible components within the Segment in addition to the segment Properties window.

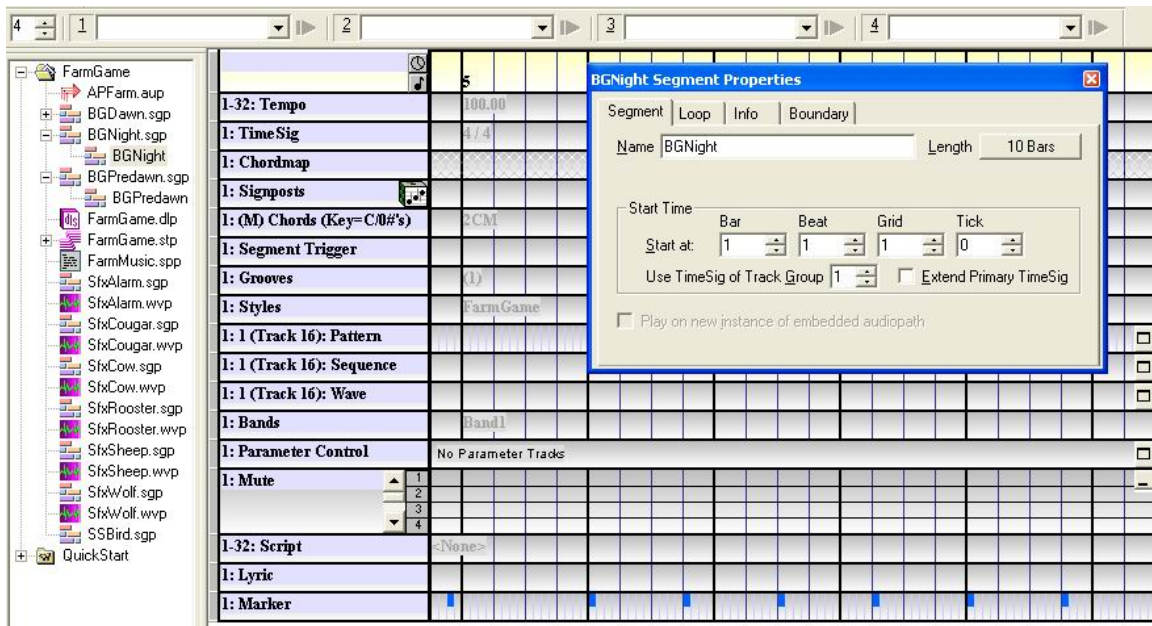


Figure 1.1 DirectMusic Producer Segment interface

All the components can be edited from this interface, providing a level of redundancy in many cases with components that are also sub-components of Styles (such as Patterns and Bands). The project folder browser on the left hand of the screen allows navigation between the components. As the top level components, Segments are used to provide overall control of the composition and bring the other components together.

1.4.3 Styles

Styles contain the main musical content for MIDI-based DirectMusic Producer composition. The three component types which make up a Style are Patterns, Motifs and Bands. Additionally global parameters can be set including tempo and time signature. Styles are important as they bring together all the features and controls necessary for MIDI scoring into one component. Figure 1.2 shows a screenshot of the Styles interface.

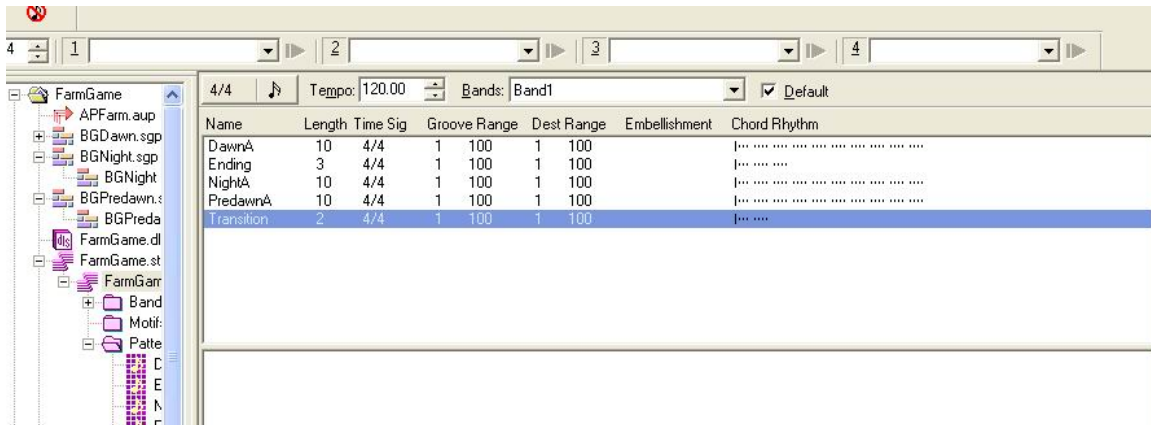


Figure 1.2 DirectMusic Producer Style interface

Information about each pattern and motif is displayed along with dropdown menus to select the default band and tempo.

1.4.4 Patterns and Motifs

Patterns and Motifs are the components which contain one or more MIDI parts which in turn contain the actual MIDI data. Patterns make up the main body of the music and are looped continuously while Motifs are used to add specific musical themes which may be triggered by in-game events although the actual construction of the two components is very similar. For example, a game character is walking down a street with a 'City' theme playing when they walk past a shop that has some significance in the game. Rather than change the whole musical score, the shop's theme is played as a Motif on top of the City theme. Using Motifs in this way is a good technique for improving the interactivity of a soundtrack. Figure 1.3 shows a screenshot of the Pattern interface.

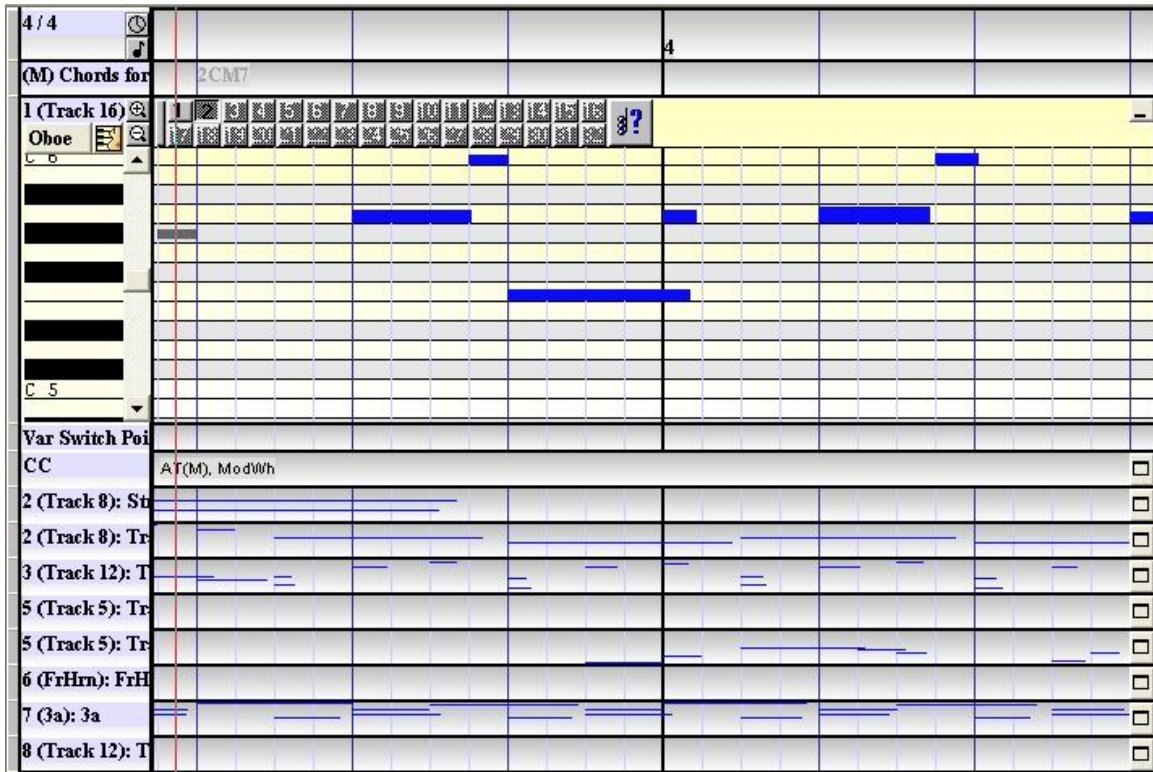


Figure 1. 3 DirectMusic Producer Pattern interface

A number of tracks are shown with the oboe part expanded out. MIDI parts (or even whole MIDI tracks) can be imported into a Pattern or MIDI data can be recorded or step-input using the MIDI editor which is similar to those found in music sequencers such as Pro Tools and Nuendo. The numbers 1-32 at the top of the Oboe track are the possible variations for that track, in this example there are 2 variations. In order to provide variety within a pattern while it is looping, a number of variations may be written for each track which can be set to random selection, play in order, no repetitions or shuffle (patterns are played through, shuffled then played again) modes. The inclusion of other algorithmic permutations here could be easily implemented; for example using $1/f$ noise (noted as being particularly musical (Dodge & Jerse, 1997)) noise as an alternative to a purely random function. Each track is assigned to a pChannel (which has the same

function as a MIDI channel except it is possible to have over a million pChannels compared to only 16 MIDI channels.

1.4.5 Bands

Bands contain information controlling which instrument is assigned to each pChannel in addition to panning, volume and transposition. Using Band components allows the composer to include changes of instrumentation and to change mix levels and panning. Figure1. 4 shows a screenshot of the Band interface.

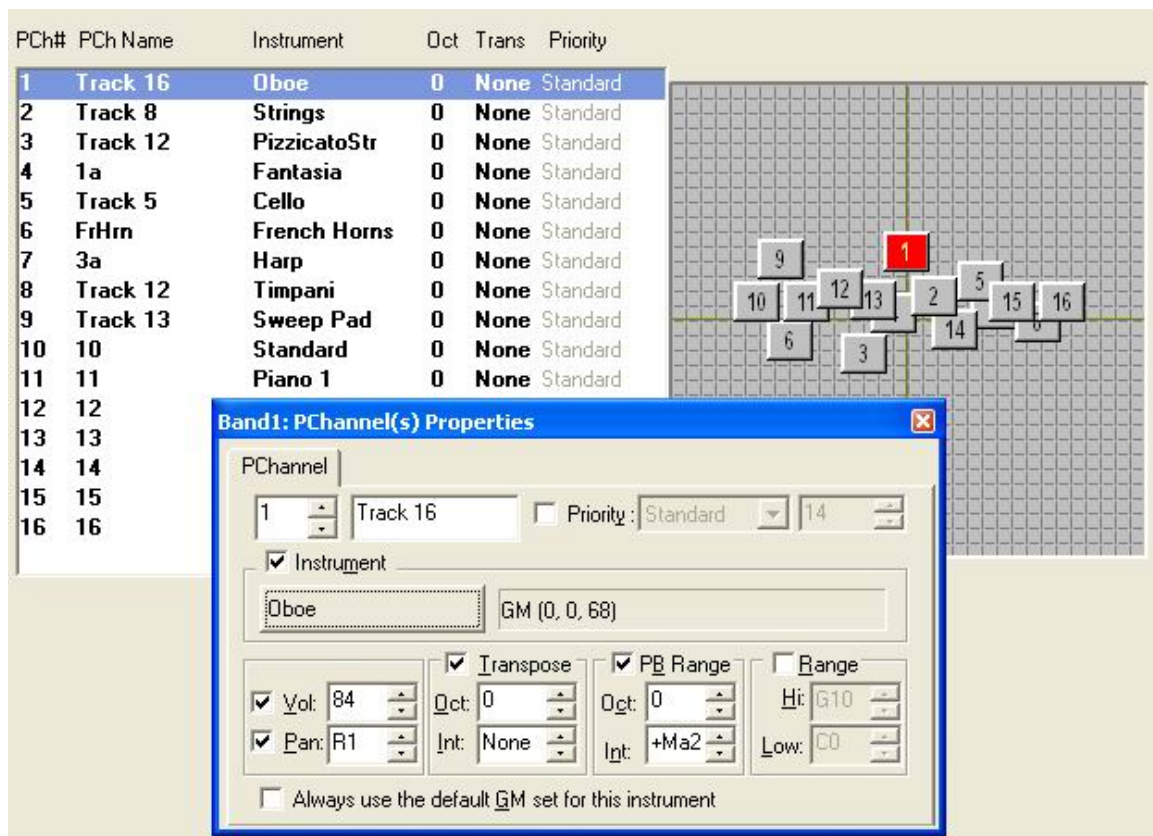


Figure 1. 4 DirectMusic Producer Bands interface

On the left, information about the pChannels is displayed and on the right is a graphical representation of the panning and volume. Channels (represented by numbered boxes) can be dragged left or right within the box to change their panning or up and down to

increase and decrease their volume. Instruments are selected in the properties box from either one of the included samples or a DLS (Down Loadable Sound) created by the composer.

1.4.6 Waves and DLS

Wave objects can be created by importing wav files to the project and can be used either as a track in a segment or sub-segment, in which case the Wave acts in the same way as a Pattern (including the ability to use a number of Waves as variations), or can be called by a script with the effect of using it as a motif. The first method is useful for creating high-quality music tracks and can be used as an alternative or in conjunction with MIDI and the second method is useful for triggering sound effects. Figure 1.5 shows a Wave object as a track in a Segment.

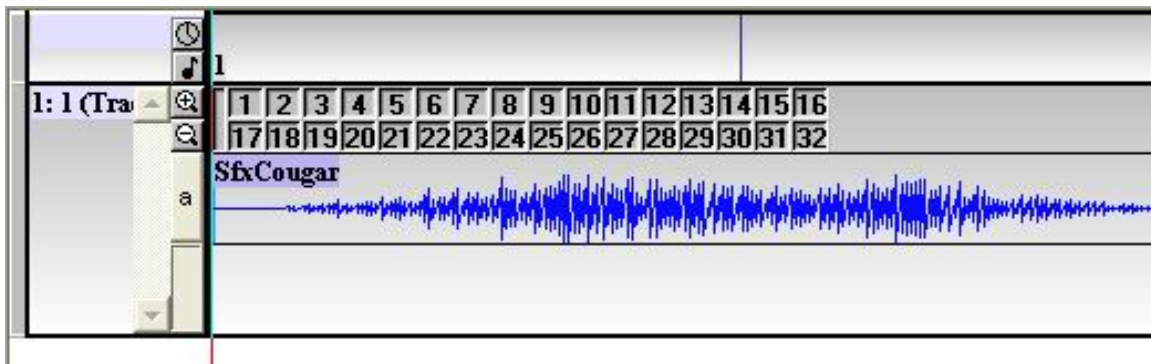


Figure 1.5 DirectMusic Producer Wave interface

There are limited wave editing features for Wave objects so it would be more appropriate to edit wav files using dedicated software such as Pro-Tools. Wave objects can also be used to make DLS objects; providing a way for composers to create their own instruments which can be used within Bands. DLS objects appear as instruments in the Bands interface and can be made from any wav file giving the composer the ability to

create new instruments, sound effects or even a longer musical theme to be triggered by MIDI notes. In order to convert a Wave to DLS, the DLS Designer window is used (Figure 1. 6) which also allows the composer to control parameters such as envelope, root key and note range.

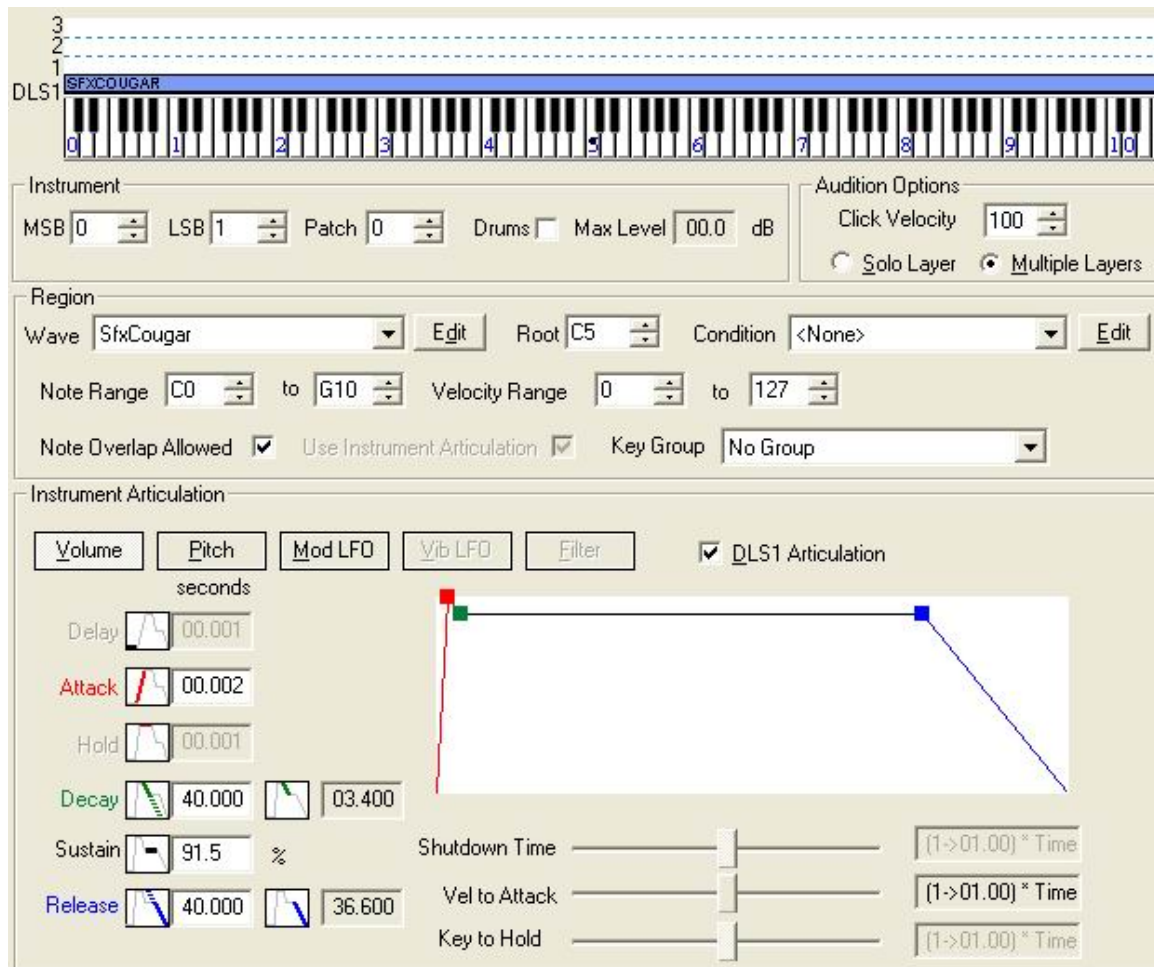


Figure 1.6 DirectMusic Producer DLS Designer window

1.4.7 Audiopaths

Sounds are mixed automatically by DirectMusic, but a composer can assign Segments and pChannels specific Audiopaths within the mixer which can in turn be configured with effects processors such as reverb, chorus and compression. Audiopaths may be

assigned statically by the composer or dynamically at run-time by using scripts. Figure 1.7 shows the interface for an Audiopath which is fed from pChannels 11 and 12 and has had a number of effects added.

Mix Group	Bus	Buffer	Effects List	Effect Palette
11-12: Mix Group-----	Stereo (L+R)	User Defined	Echo1 Compressor1 Chorus1 I3DL2Reve	Chorus Compressor Distortion Echo FileOutput Flanger Gargle I3DL2Reverb ParamEq WavesReverb

Figure 1.7 DirectMusic Producer Audiopath interface

1.4.8 Transitions and Markers

As one of the most important aspects of interactive composition, transitions form an integral part of the DirectMusic Producer system and involves two key parameters controlled by the composer. The first is to create a suitable Pattern to be used as a transition and assign one or more embellishments which may be intro, end, fill, break or custom as shown in Figure 1.8.

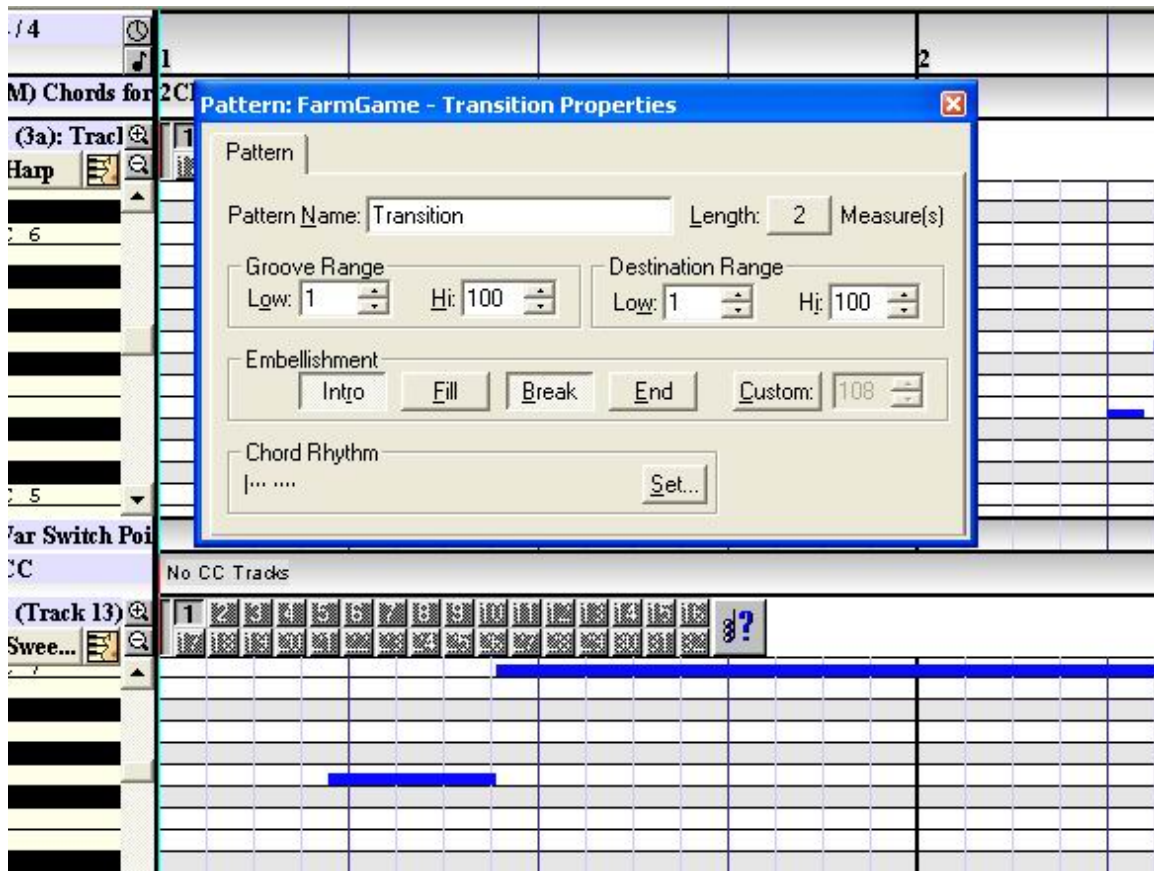


Figure 1.8 Adding embellishments to a pattern in DirectMusic Producer

The embellishment tells DirectMusic producer what type of transition the pattern is and will effect how the music modulates between the Patterns. Intro transitions will play based the pattern they are introducing, end transitions play based on the pattern they follow and fill, break and custom transitions will modulate between the two. It is also important to set up a Chord track for each pattern specifying what chords the pattern is based on so that DirectMusic can make appropriate modulations (Figure 1.9).

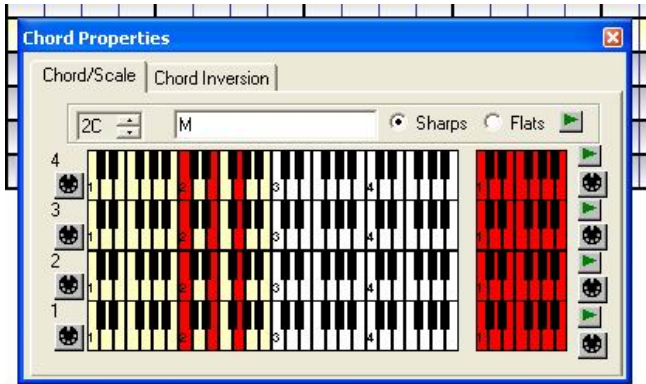


Figure 1.9 DirectMusic Producer Chord properties

The second measure of control is used to restrict the points at which a transition can occur. In a pattern it is likely that there are certain points at which a transition would be more musical than others and it is just as likely that in an interactive situation, transitions will not be triggered at these points. To stop this from occurring, transitions are constrained to occur only at specific points known as markers. A screenshot of the marker track is shown in Figure 1.10.



Figure 1.10 DirectMusic Producer Marker track

The composer can add both transition out (markers in the top half of the track) and transition in (markers in the bottom half of the track) to indicate points which are acceptable for transitions.

With the marker and embellishment system it is possible to compose a score which will flow smoothly from any one pattern to another although it relies heavily on the composer to write patterns and transitions which work well and have a number of points in the

music which are suitable for in/out markers. This can be difficult and often restricts the composer's creativity.

1.5 Game Audio in the Future

It is clear that some significant steps have already been taken toward improving Game Audio. A variety of techniques such as 3D sound design are used in the majority of current games to provide realistic audio effects. DirectMusic Producer has provided games composers with a free music sequencer which is not only designed for interactive composition, but works on a very close level to the code which programmers will use to implement audio in games. New technology in the next generation consoles will provide audio programmers, composers and sound designers with far less limitations than ever before and at the same time are capable of implementing 5.1 surround sound; opening up possibilities for audio to become much more important to gameplay,

I think both. I think we've barely scratched the surface of what is possible with, for instance, interactive music in games. The hardware is getting better and better as well, supporting all sorts of surround formats from the get-go.
[de Man, 2005]

Already many games designers have noted the potential to use sounds not only as an indicator of events happening off screen, but also as a key gaming element. A good example is the upcoming game Metronome in which the player has to record in game sounds and use them to interact with the world,

"It wasn't that we said, 'Wow! Let's make a game with sounds'. It's more that we started off with the story, and as it progressed we found that sound was a key element." As development progressed, the team found that this focus on recording sounds within the game completely changed the way players interact with the world... the ability to record sounds means players can demonstrate a certain amount of creativity. They can make their own sounds by throwing a rock through a window
[Jordan, 2005: 33]

The next stage of development for game audio is to concentrate on using the emotive quality of music to immerse the gamer in the experience. Many game audio designers consider Films to be the best role model for immersive and emotionally involving soundtracks (Morton, 2005a) and convergence between the two media is prolific. It is also commonly felt, however, that the challenges of composing interactive music are prohibitive towards achieving that goal. The difficulty of transferring concepts of style from a linear medium to one where a score may need to transition at any point according to the player's actions is well known.

From these points we can conclude that the ideal game soundtrack would consist of film-like composition and sound design but would be able to creatively apply complicated transitions between segments even in the middle of tracks and between tracks that have very little in common musically. Another important feature would be that the score could react appropriately to the events in the game and provide score that worked to enhance the emotion even in complex situations and states where a simple trigger system might fail to provide an appropriate soundtrack. One potential solution would be to use artificially intelligent agents to attempt to mimic the behaviour of the human composer and/or sound designer in real-time within the game. The first step in such a process is to analyse the methods of human composers; in this case those associated with film.

Chapter 2

Film Audio

2.1 The convergence of Film and Games

Film music and sound design have had a huge impact on computer games; with an increasing demand on computer games composers to deliver a 'Hollywood-style' soundtrack (Stevens, 2001). Over the years there has been significant crossover between the film and game genres with Hollywood movies often being licensed as computer games and occasionally vice-versa (e.g. *Resident Evil*). In the past this has consisted simply of building a game which follows the plot of the film, placing the player in the role of the main character with little innovation in terms of gameplay or storyline. Recently, however, some companies have been exploring the possibilities of film/game crossover more thoroughly, most prominently the Wachowski Brothers with the Matrix series:

Our films were never intended for a passive audience...Gaming engages your mind actively whereas most genre films (the films we tend to watch) are designed to provoke as little thinking as possible...For us, the idea of watching our baby evolve inside the virtual bubble-world of this new radically developing medium, which has in our opinion the potential of combining the best attributes of films and games...

[Wachowski Brothers, 2005]

To complement their film trilogy, *The Matrix*, the Wachowski Brothers made two computer games. The first game, *Enter the Matrix*, is set parallel to the second film in the series and – importantly – complements the plot of the film rather than simply repeating it; demonstrating an 'unprecedented level of collaboration' between the film makers and game designers (www.enterthematrixgame.com) . The second game, *Matrix Online*, is

set after the final film in the trilogy and is a massively multiplayer online game (MMOG), allowing thousands of people to simultaneously play within a simulated world and take part in the continuation of the story. In this example, the trademark sound design of the Matrix has been integrated within the game including voice-overs by the original actors.

Games design company Square Enix has also been working on film/game integration and in autumn 2005 plan to release *Advent Children*, a digital film which continues the story from their hit game *Final Fantasy VII*, with the trailers featuring orchestrated music from the original game.

In addition to direct crossovers between the film and game industries, obvious influences from the film genre can be seen in games which feature Hollywood actors and composers, such as *Onimusha 3* which stars Jean Reno playing a lead character and *Sun* which includes score by Howard Shore. At the 2005 Game Developers Conference in San Jose, a lecture by Jared Emerson-Johnson divided modern computer game music into five categories according to their influence from film and classical composers:

The "Hans Zimmer School," a hard, driving style, typified by parts of Metal Gear Solid 3: Snake Eater and Chronicles of Riddick... the "Danny Elfman School," for the sake of clarity: an alternative, quirky, playful, dark, subversive style with a strong 1-5 "oompah" progression in the bass line... Carl Orff's "Carmina Burana": as Emerson-Johnson described it, "epic choral primitivism, undeniably hard-core;"... "The Planets," by Gustav Holst: "driving, solid, intense, serene but dominating;" Emerson-Johnson played a clip from the upcoming God of War, for comparison. Igor Stravinsky's "The Rite of Spring": what Emerson-Johnson described as "Danny Elfman through a John Williams filter." He made note of Stravinsky's "orchestral primitivism, formal schizophrenia, [and] meticulous orchestration," and referenced KOTOR 2: The Sixth Lords.
[Waugh, 2005]

To understand what makes film sound design and music so relevant to computer games, it is necessary to consider what the similarities and differences between the two media

are and why it can be desirable to implement techniques used in film audio when designing computer games.

2.1.1 Sound with picture

The most obvious parallel between film and computer game sound design is that both involve the integration sound and video. In both films and games it is important to build a believable soundscape, which complements what is happening on screen and draws the audience/player into the experience rather than distracting them from it. There are two main categories of sound design, synchronous and non-synchronous.

The use of synchronous sound design and music was proliferated in the early years of sound cinema (Weis & Belton, 1985) and remains the primary form of sound expression in games. Synchronous sound design involves a close link between the picture and sound, with the sound often viewed as subservient to the picture. In interactive media this also extends to off-screen sounds which are synchronized to objects or actions. The reason for this elaboration is that in an interactive product there is usually not one picture at any one time but many possible pictures as the player may change the perspective of the character. Some games with a third person perspective have fixed camera angles in each scene (usually to allow the use of pre-rendered environments) and for these the concept of synchronous sound is the same as for films. Music is considered synchronous when it closely follows the actions and emotions of characters on screen particularly when music is used to underscore important visual events (known as 'hitting').

As sound cinema progressed, some directors such as Pudovkin and Claire (Weis & Belton (eds), 1985), questioned the subservience of sound to picture, and began to explore the creative possibilities of asynchronous sound design. The theory of asynchronous audio places sound on the same level of importance within films as

picture; promoting the abstraction of sound from picture as a method of portraying a deeper level of understanding to the audience or to provide contrast and contradiction (Ibid). Although a connection is retained between the sound and image, it is not the naturalistic connection between an object and the sound that it makes or between the music and the emotional content of the scene. Although asynchronous sound design and composition is a well known and frequently used tool in film audio, it is rarely seen in game audio where it is more difficult to implement. The decision to abstract sound from picture is an inherently creative one and must be accompanied by appropriate technical expertise, however due to the interactive nature of games; sound design and composition must be performed in real time by a computer program.

Voice (and dialogue) may be implemented synchronously or asynchronously, but in either respect is considered separately from other aspects of sound design as it primarily provokes an intellectual rather than emotional response from the audience (although a secondary emotional response may occur; for example in the case of a stirring speech).

2.1.2 Telling a story

Another common theme between films and computer games is the story or plot. While all films have a plot, however, there are game genres that are not based around a story, such as sport and racing simulations which primarily provide entertainment out of a sense of challenge and competition or even socially when playing with friends (Kane, 2004). Games such as these have few parallels with the plot based audio and are discounted when comparing films and games regarding plot.

The plot encompasses the bigger picture, sets the overall context, style and structure of either film or game. In films this is very specific, being a linear and rigidly set journey, games on the other hand may have a looser plot or even several parallel plot lines depending on choices made by the player as is seen in *The Suffering* by Midway. Other

games, such as *Morrowind* provide the player with a great deal of free choice, to the point where the player may choose whether or not they actually follow the plot at all. For audio to achieve its full potential in terms of immersing the audience emotionally, it must serve to aid the development of the plot and be tied in very closely to its progression. The very nature of what is generally perceived as well written music involves the concept of structure and development, strongly lending itself to plot based roles. Sound design which does not serve a musical purpose in the short term² may develop a musical quality through virtue of its close association with the plot. Films often achieve excellent results in this respect, making good creative use of both music and sound design. In comparison, although game audio often makes use of basic plot based development in audio (e.g. the level music system), the same level of creativity and detail has not been reached in games.

2.1.3 Audience

Franchises, such as the Matrix series, have released successful products in both film and game genres and a major selling feature of current and next-generation games consoles is the capacity to play DVD films. This indicates strongly that there is a significant shared audience between the two media; consideration for the intended audience is important in the creation of any artistic work, particularly if it is to be a commercial success.

² The distinction between musical and non-musical sound design can be confusing but is an important one to make; sound effects which are specifically designed as part of the score are carefully selected as musical elements and are considered part of the composition. Typically musical sound design is performed by a composer and non-musical sound design (usually referred to simply as sound design) is performed by a sound designer.

2.1.4 Interactivity

By far the most significant difference between films and games, while a film is the same every time is performed³, a game depends entirely on the actions of the player for its realisation. This presents several challenges to game sound designers and composers (as described in chapter 1); however it also presents many opportunities. Games have the potential to be more emotionally, morally and intellectually immersive than film ever can, but in order to realise that potential there must be a certain amount of innovation regarding interactivity. Films have gained a degree of interactivity with the popularisation of DVD's which use interactive menus to navigate scenes and access special features which may include small games and even alternative endings. The interactivity, however, is still very peripheral to the film itself (which is often called the main feature).

2.1.5 Length

The second biggest difference between films and games is the length. A film will typically last between two and three hours but it is not unusual for a plot based game (such as an RPG) to boast ten times that amount or more. A new game genre, MMORPG (Massively Multiplayer Online Role-Playing Game), which is fast growing in popularity provides a persistent online world for the gamer to play in and – in theory – never ends.

2.1.6 Comparison: Film and Game

There are many parallels between the film and game media. With its long history of using sound and picture to express drama, film audio provides an excellent role model for creatively using sound and music to immerse a player within a game. Audio in games

³ The presentation of the media is intended to be the same every time, although it may be argued that the experience of a film can be different watching in different places or over several viewings, this does not pose the same challenges faced by interactive media.

is often given a similar status as audio in the early sound cinema, by learning from the evolution of film audio and applying similar concepts to games, we can potentially make rapid advances in the quality of the gaming experience. Much of the intended audience for games is also part of the audience for film, strengthening the case for creating film-like soundtracks within games. Because of the significant differences in interactivity and length between films and games, it is not a matter of simply writing film score for games and a new approach must be taken which allows techniques from film sound design and composition to be applied to long-duration interactive drama. For some genres, however, such as sport and racing simulations, a film like score is not appropriate. Audio Visual Example 5 shows *Gran Turismo 3 A-Spec* a racing game where the music is a fast paced dance track and synchronous sounds are used for engine noise and screeching tyres. At 35 seconds, the music finishes and a new track begins to play; in drama and plot driven games this form of transition is inappropriate as is using linear music as a backing track, however in a racing game such as *Gran Turismo 3 A-Spec*, the music is both appropriate and effective at communicating a feeling of energy and speed.

2.2 Film audio techniques

Audio was first introduced to the film industry in the late 1920's and, although at first meeting strong opposition from many directors of silent film, has become an integral part of the modern cinematic experience. Early film sound theory considered sound to be subordinate to the image, with both sound effects and music closely following the picture,

From the very first critics who had lived through the coming of sound took the historical process (whereby an art which had once lacked sound had the capabilities of sound reproduction added to it) as an adequate model for theoretical

reflection. Instead of treating sound and image as coexistent, the historical fallacy orders them chronologically, thus implicitly hierarchizing them...With the rapid universalisation of sound technology, however, the force of historical argument necessarily subsided.

[Weis & Belton (eds), 1985: 51]

As cinema progressed directors became more aware of the fundamental role that sound was playing in cinemas and of the expressive opportunities of sound. While some critics still argue that picture is the primary form of expression in films, we only have to consider the fact that the first step in making a film is the script which focuses on the dialogue – a sound element – to see that this is not the case in practice. One reason that sound is often labelled as less important than picture is that the effect and potency of sound is very hard to analyze and articulate; our understanding of sound occurs not on a rational level but on an emotional one. When we experience sound, we do not perceive it as an object in its own right but as a property of another object; for example if we hear a knocking sound, we do not understand it as a 'knock' but, perhaps, as the result of someone striking the door with their knuckles. What is more we understand that this means that there is someone outside the door who desires to be let in. Chion (1994) termed the introduction of a sound without an image of its source as 'acoustimatic sound' and although at first it may seem to support the concept of sound as the subordinate of image, the ability of sound to conjure imply the existence of objects, actions and even emotions is what makes it useful creatively. As cinema progressed, filmmakers began to capitalize on this through use of asynchronous sound either to work in parallel with the visuals to create a sense of realism, or to contradict the picture, providing contrast. Filmmaker theorist Robert Bresson saw sound as a way to liberate the image by replacing it, believing that image and sound have equal and independent roles in cinema:

A sound must never come to the help of an image, nor an image to the help of a sound...If a sound is an obligatory complement of an image, give preponderance

either to the sound or the image. If equal, they damage or kill each other...Image and sound must not support each other but must work each in turn through a sort of relay.

[Weis & Belton (eds), 1985: 149]

Bresson identified the fact that nature of our interaction with sound lends itself to providing us with different knowledge than we gain through our interaction with sight and that by using the two senses independently, the best result is achieved.

Game audio theory is very backwards in this respect and, while acknowledging the importance of sound, still places a similar emphasis on the picture as was seen in early cinema. Recent advances in gaming technology, including surround sound presentation of audio, have opened up the possibility of a large leap forward by learning from the experiences of film sound design, composition and theory. As the games industry progresses, it will be vital to understand the different roles that picture and sound perform in providing the player with knowledge; not only of the game environment, but also of the plot and the emotions of the character being played. Among the most important aspects of film audio to be considered are the use of synchronous audio, asynchronous audio, silence, association, transition and structure. It is not enough to simply apply the linear methods and techniques used in film to games; they must be understood and adapted for the interactive medium.

2.2.1 Synchronous sound

Sound effects in synchronous styling are used to 'reveal the acoustic landscape' (Weis & Belton (eds), 1985), re-creating the physical sounds of objects in the scene. Some sounds are used routinely to immerse the audience into the world of the character with background sounds such as traffic and also with Foley, which is a term used to represent sounds that the character(s) make with their hands and feet. By placing the audience in

the same sonic environment as the characters, they are drawn into their world and will more readily identify with the characters' perspectives. Games primarily make use of synchronous audio, owing partly to the fact that it is programmers and not composers or sound designers who implement the run-time systems that control sound design within the game.

A more creative use of synchronous sound, which can be particularly effective, is to place more emphasis on the sound of an object than is natural; for example raising the volume of a ticking clock emphasises the passage of time. One of the abilities of the human hearing system is to focus on individual sounds when presented with complex soundscapes; by raising the level of a sound above others in the mix, the audience is not only made aware of the sound (and therefore the object or action it belongs to) but understands that the character or characters are focussing on the object, which in turn places them in the perspective of the character. In terms of gameplay this can be used to communicate to the player the thoughts and emotions of the character. If there is something that the character is meant to notice but which the player might not, a traditional solution would be to perform a cut scene (where the player relinquishes control of the character) in which the character is seen to notice the object or to use an on screen marker. The problem with both these methods is that they act to remove the player from a feeling of immersion:

If I've been playing a game for an hour or so, I've learned to identify with the character I'm controlling. It goes beyond identification with a film character.. it's an extension of me. When we are talking with friends about our recent Halo escapades, we don't say, "It was cool when Master Chief lobbed that grenade from the cliffs and took out a Warthog." Instead, we say, "It was cool when I lobbed that grenade." Even if Master Chief doesn't look like me or move like me, I'm still attached because I'm in control. I'm actively participating in the character's fate. Then comes the cut scene. Suddenly I'm not in control of my character anymore. [Morton, 2005a]

Using sound to focus in on an object not only avoids the need for a cut scene, but actively promotes a feeling of immersion as the player feels they are experiencing their character's senses.

Although many film theorists look down upon the use of synchronised audio, there is a strong case for including it within game sound design. Because games are interactive, it is usually necessary for the gamer to feel that their actions have an impact on the game world and an effective way of achieving this is using sound, particularly as that sound will be able to strongly influence the player's emotional interaction with the game. Also since synchronism in sound favours a naturalistic view of sound production, it can be very useful when presenting the audience or player with something which is not in their natural experience. Audio/Visual Example 6 shows a scene from *Star Wars Episode II: attack of the clones*. In this scene the audience is presented with a number of unnatural phenomenon, in particular a monstrous creature and a lightsaber. The sound of the creature's cries help to give the creature a more believable sense of reality while the distinctive hum of the lightsaber communicates the power of the weapon to the audience.

Music varies in games between being highly synchronous in its reactionary form and highly asynchronous in its ambient form. As with films, supporting the action on screen with music can deepen the emotional involvement of the character. Synchronous music may also be used to represent in-game sources such as radios, juke boxes and musical acts. Rockstar's *Grand Theft Auto: San Andreas* made effective use of this by providing musical soundtrack via a tuneable in-car radio system (Greeves, 2005).

2.2.2 Asynchronous sound

Asynchronous sound design involves abstracting the audio from the image and can be a particularly powerful creative tool. Abstracted sound is used in parallel to the image, i.e.

although not married to the image both are working towards the same goal, may involve including the sounds of objects or actions which the audience cannot see and usually functions to enhance the audience's perception of the world and communicate extra information. By using audio in this way, the sound designer is taking greater advantage of Gestalt psychology which states that we form our perception of the world through knowledge gained from the five senses but that the result is greater than the sum of the parts since knowledge from one sense influences the perception of knowledge from each other sense (Sonneschien, 2001). Games in general do not make extensive use of asynchronous sound design and music, but do sometimes include it in the form of ambient soundtracks which set the mood for a level without representing onscreen action; for example in a battle themed level there may be shouts and sound of fighting even when there are not any characters on the battlefield. Unfortunately the implementation of this is often neither particularly creative nor reactive.

Asynchronous audio may also be used in contrast and contradiction to the picture and in this form can be seen to elicit the strongest emotional response since each sense will influence the perception of the other. When presented with contradicting senses, the brain will attempt to resolve the conflict by generating new knowledge that establishes a rational connection, and in doing so involves emotional reasoning at a much deeper level than is accessed by either sound or image alone. A common implementation of this is setting peaceful or calming music to a violent battle scene as shown in Audio Visual Example 7, a scene from *Gladiator* where the central character, Maximus, has fought his way through the lines of his enemies, As Maximus realises that his forces are winning the battle, the sounds of battle die out and are replaced by calm music. Through this the audience understands Maximus' relief and that for him the battle is over. An example of a similar event using sound design is seen at the beginning of *Apocalypse Now* where

careful abstraction of the ambient sounds is used to communicate the desire of the main character, Captain Willard, to be back in the jungle and away from the city:

At the very beginning of the film, Captain Willard (Martin Sheen) is in a hotel room in Saigon... Gradually what happens is that all of those street sounds turn into jungle sounds: the whistle of the policeman turns into a cricket; the car horns turn into different kinds of birds; and the fly turns into a mosquito. You are watching Willard sitting in his hotel room, but what you are hearing is a very strong jungle background. One reality is exchanged for another. The thread that links them is the fact that although his body is in Saigon, his mind is in the Jungle.
[Weis & Belton (eds), 1985: 356]

The challenge with asynchronous audio is in using it creatively and appropriately: creating the wrong emotional response is easily achieved or even worse the audience may become confused. In a film, the composer or sound designer may carefully orchestrate the audio to have the desired effect; humans working offline with a full knowledge of the context of the scene. In a game, a computer working in real-time is responsible for orchestrating music and sound. If game audio is to feature asynchronous sound design and music, an advanced technique of selecting the appropriate sounds and/or music at the appropriate time will be necessary.

2.2.3 Silence

One of the benefits of having sound in a game or film is that we then have the artistic choice not to use it. Silence can be a powerful tool when used effectively and is in fact a special case of asynchronous sound design. Audiences have become so used to having music and sound in films and games that they find it unusual not to hear it. By including moments of silence, a stark contrast is made with the sounds which by comparison seem louder and gain significance. Sonnenschien (2001) describes a scene from *Face to Face* which uses a clock sound in isolation to communicate the incredible loneliness of being alone, the stark contrast of the ticking emphasising the anxiety of the character

in the scene who is a profoundly depressed woman. Instead of using music to influence the audience's emotions, they are encouraged to empathise with the woman by hearing the focus of her mind on the clock. A similar situation occurs in *American Psycho* at the end of the film, where the main character Patrick Bateman, who has during the course of the film killed a number of people, believes he has been caught and confesses over the phone to his lawyer (Audio Visual Example 8). Throughout the film, music has featured strongly and Bateman is a fanatical music fan; the absence of music during his confession portrays Bateman's feeling of isolation and helplessness.

Silence can be less difficult to implement in computer games than other forms of asynchronous audio and several survival horror games, such as the *Resident Evil* series, make good use of silence contrasted by sudden loud sounds to shock the player. Many games however do not make any use of silence, be it creative or mundane. Films will rarely have music playing throughout as this diminishes the emotional impact through a lack of contrast. Many games on the other hand will have music playing constantly throughout. Although this may be appropriate in racing, sports and action games, it is counter-productive to providing good drama.

2.2.4 Association

As previously mentioned, we perceive sound as an association rather than an object in its own right. This is not limited to perceiving sounds as being associated with a physical object, however, sounds may also be associated with feelings, emotions and other associations in various complex ways. For example a punch on screen may be accompanied by a slap and or a thud noise, which is associated with the impact of the punch. The loudness of the sound will be associated with the force behind the punch and the tonal characteristics will be associated with the damage received – more slap indicates a glancing hit, more thud will be interpreted as a penetrating blow. In reality the

punch was staged and probably never even connected. Often effects such as this are not intended to sound realistic but are exaggerated. Another example of association is a boulder that makes a grinding sound as it is moved. The grinding sound is associated with the friction resulting from the weight of the boulder; therefore the audience understands that it is a heavy boulder even though it is really made out of polystyrene. According to Sonnenschien (Ibid) sound may be classed into four types of association: physical, emotional, intellectual and moral. Physical associations may be used to communicate the properties and attributes of objects such as the weight of a boulder and are usually related to the physical laws which would produce the sound. Emotional associations are used to influence how the audience feels and the sounds are usually related to the perspective of the character; although a real punch rarely creates a loud thud, the feeling of a connecting punch sends vibrations through the bodies of both puncher and punchee which, from their perspective sound very much like a thud. Intellectual associations are primarily communicated through language, although other forms of communication such as Morse code may also have intellectual associations. Vocal noises which are not part of language may be associated with either the intellectual or emotional or both depending on the context and intention of the sound. Moral associations are usually concerned with human choice and therefore also the plot and as such are represented by structure and development within sound and music. Given that much of human choice is based around our emotional state, moral and emotional associations are often closely linked.

Music also has several associative properties intrinsic to its nature. Certain chords, keys and intervals have associations with a variety of emotional states; the most obvious example being major (happy) keys versus minor (sad) keys. Proper use of these associations is vital in producing an emotionally involving and appropriate soundtrack. Additionally music often has social associations such as historical period and location.

Matching the musical instrumentation and/or style of a period or location helps the audience to identify with the film or game and become immersed in the experience.

2.2.5 Transition

Transitions are usually associated with the musical score of a film or game but also are important in sound design which, in terms of transitions can be thought of as a type of music. As a film or game progresses, the music and sound must change to follow the course of events and transitions are the techniques used to do this. Traditionally linear music has a wide range of transitional techniques which are used to creatively develop and progress the piece. Film sound also uses transitions in this way but with the added constraint that of following the plot and events within the film. Game audio uses transitions purely to follow the actions of the player allowing for changes in music to occur to reflect the emotional state of the character or for the sound design reflecting changes in the environment. Typically transitions in games are simply seen as a method of moving between two pieces of music or sound with little application of creativity; game music composers are encouraged to write music which will seamlessly transition to all other pieces of music. This is due to the challenge of providing an interactive score.

Musically there are a number of ways to make a transition with changes in tempo, key and transposition being popular but also including dynamics and instrumentation. In fact any musical variable may be used to progress a transition and transitions can occur slowly, gradually evolving or change instantly to follow the fast action of a scene. If transitions are overused, however, the continuity of a scene can be compromised; a sense of unity should be kept throughout (Rona, 2000). Techniques such as maintaining a constant tempo, theme or instrument part across a number of transitions can be used to retain a sense of continuity.

Solving the problem of performing creative transitions within game audio which requires freedom to respond interactively to the player's actions is vital to advancing the quality of interactive music. A system capable of this would necessarily be able anticipate the player's actions to some extent in order to plan ahead of the transition and retain continuity.

2.2.6 Structure

Inherent in any piece of music is a structure or form which organises the development of the piece both in context of both the entire piece (macrostructure) and each section of music (microstructure). Commonly both macro and microstructures will include elements of self-repetition and variation, extremes of either are usually thought of as either dull and monotonous (repetition) or chaotic and random. The macrostructure and microstructure are also commonly linked as observed by Dodge & Jerse (1997: 369): 'Self similarity is characteristic of much traditional music where the local detail mirrors the structure of the overall piece'. Repetition plays a key role in structure; it is through use of repetition that an audience will understand and become familiar with the structure of the music and by contrasting the repetition with elements of evolution and variation that the progression of story and plot is expressed and a sense of drama created,

Through the repetitiveness of a phrase (for example Bach's "Canon in D", Ravel's "Bolero", or Philip Glass works), listeners can become familiar with it and comfortable. In doing so, they will become more receptive to other inputs on an emotional level. However, if the phrase repeats identically like a mantra, this can either draw the listeners totally inward and away from the external storytelling...or drive them away from the film because of boredom and distraction.
[Sonnenschien, 2001: 116]

Historically, an important dramatic musical structure has been the Sonata form (Rosen, 1988). Consisting of three stages, exposition development and recapitulation, Sonata

form begins with two thematic ideas which are contrasted and altered before being repeated again. The reason that Sonata form is so often used for dramatic composition is that its structure mirrors the typical structure of a drama: at the beginning there are one or more characters that have a goal, during the drama they face a number of challenges and at the end there is resolution, either through the character(s) accomplishing or failing at their goal. Both macrostructure and microstructure may benefit from Sonata form and, if used in both, an additional benefit of self similarity is achieved.

For film and game audio, macrostructure is tied into the progression of the plot. In films this is usually considered to be the single most important factor in composition and sound design (Rona, 2000), however in games it is often sacrificed in favour of a more reactive score/environment. Two methods of providing a more plot based structure which may be adapted into non linear forms for use in games are themes and sound maps.

Themes are often considered obligatory in modern film music and may be found in the form of a general theme which encompasses the entire concept of the plot or more specific themes (known as leitmotifs) which represent individual characters and concepts,

Every film score needs a primary thematic element that makes it unique, identifiable and memorable. In most cases, the thematic element is the main element used to introduce the score. There are many well-known film scores that are identifiable by their main musical theme. That theme can be a full blown, melodic idea, like the scores of John Williams, a consummate melodist. Some melodic themes can be small fragments, such as the famous shower scene from Bernard Hermann's score to Psycho (arguably the most famous film theme in history), or William's Jaws...Every film score needs a "door", a single, unique way in to help give it focus and identity.

[Rona, 2000: 3]

Writing music with themes in is one of the simplest and most effective ways of engaging the audience and promoting the plot and is equally applicable to sound design which

may use repetition of a particularly identifiable sound or combination of sounds (e.g. Ben Burt's use of the distinctive lightsabre sound in the Star Wars series). Writing music with themes is no more complicated for games than it is in films; the challenge arises with respect to the development of themes throughout the game and particularly in accomplishing interactive thematic development when sections of music are either selected randomly or triggered in reaction to specific events. A possible solution is to conceptualize the plot in terms of bi-polar extremes.

Sonnenschien (2001) describes a method of using opposites of dramatic themes within the plot to create sound-maps which describe the plot in terms which can be more easily translated into musical concepts. The first step in creating sound-maps is to identify bi-polar extremes within the plot. Examples of common extremes are death-life, power-weakness and good-evil. Plot elements, themes and/or characters are then given an association with one or more of the extremes. In a plot where the hero was trying to prevent a villain from killing lots of people, the hero would be associated with life and the villain with death. Interactivity lets games take this a step further by including choices made by the player in the same way. As the storyline develops, the plot will move between these dramatic extremes. In film a sound map may be drawn up where the position of the plot in terms of time and dramatic extremes is plotted on a graph. The composer or sound designer can then see how the plot develops and provide score or audio which supports that development with its own structure. In games it is impossible to design sound maps in terms of a fixed chronological scale and a game plot may have more than one possible storyline. An adaptation of sound maps would be to use variables to keep a score of the current plot position and have the real-time composition program choose thematic development based on these scores.

2.3 Implementing film audio techniques in games

We have seen that there are a number of techniques developed for film audio which could enrich the experience of game audio if properly applied. Concepts such as the appropriate use of asynchronous sounds effects and music can heighten the emotional involvement of the player while the use of structure can improve the perceived quality of music while promoting development of the plot. Transitions form an important aspect of audio in both films and games, although games rarely realise the creative potential of transitions. Fundamental to the concept of audio is their natures of association to objects and emotions and by using these associations, sound can strongly influence a player's perception of events, particularly with regards to the emotions and feelings of characters.

The interactive nature of games allows them the potential for greater immersion of the audience (player) than other media, it presents challenges for audio and sound design in that the composer and sound designer lose control of part of the creative process when the composition is realised in real time, making it difficult to use advanced techniques such as those designed above. Audio is not the only aspect of games to face this challenge, however; in films the actions of characters are carefully co-ordinated by the director but in a game the characters must react to the actions of the player. The solution which has been implemented in provide in game characters with realistic reactions to the player, their surroundings and each other is to use artificial intelligence as seen in The Elder Scrolls IV: Oblivion,

Oblivion features a groundbreaking new AI system, called Radiant AI, which gives non-player characters (NPCs) the ability to make their own choices based on the world around them. They'll decide where to eat or who to talk to and what they'll say. They'll sleep, go to church, and even steal items, all based on their individual characteristics. Full facial animations and lip-synching, combined with full speech for all dialog, allows NPCs to come to life like never before.
[Rockville, 2004]

With a precedent already set for AI being used to solve the artistic challenges posed by interactivity in gameplay, a logical next step is to propose using AI as a solution to the challenges faced in audio.

Chapter 3

Using AI for Interactive Audio

3.1 What is AI?

Artificial intelligence is a branch of computer science which is dedicated to the methods and design of computer systems which can perform tasks which would traditionally require human intelligence to perform. Paradoxically, many tasks which humans find very difficult, such as performing complex mathematical operations even very large numbers, computers excel at but tasks which we find incredibly simple (so simple in fact we do them without conscious awareness of the task) such as recognising a face, computers find very difficult (Cawsey, 1998). There are many approaches used to tackle the challenge of AI, each with a focus on specific areas of problem solving. All approaches, however, usually fall into one of two design philosophies; programs which attempt to think or act like humans and programs which attempt to think and/or act rationally.

The philosophies of designing AI which simulate human intelligence or behaviour (known as the top-down approach) assume that since humans have the most highly developed intelligent reasoning in our experience, the best way of designing an intelligent program (or agent) is to imitate the thought processes or actions of a human. Top down design is often used to create agents with elaborate problem solving mechanisms which are flexible although may not always provide the best solution to a problem.

Designing AI with rational problem solving strategies (or bottom up approach) assumes that although humans have a highly developed intelligence, it is also highly generalized, and by designing models for intelligent reasoning from scratch it is possible to create more efficient agents for specific tasks. Rational agents often excel at expert tasks which require a small amount of highly specific knowledge but are extremely inflexible.

Despite the differences between the two approaches, there are some features and techniques which commute between the two. All AI agents must be able to store knowledge and then apply that knowledge in the correct context to solve a problem or perform an action. For all but the simplest problems, an agent will require a significant amount of knowledge to be able to act correctly; particularly when interfacing with the real or a simulated world. Because of this, a number of methods for searching knowledge have been developed to help agents find a solution within a reasonable amount of time. These methods include alpha-beta pruning, A* and Heuristic searching (Ibid).

Laird & van Lent (1999) propose that an AI engine for use in computer games can be considered in terms of three primary components: the inference machine, interface and knowledge base.

3.1.1 Inference machine

The inference machine is the central component of the AI engine design because it sets for the constraints that the other components must meet. The job of the inference machine is to apply knowledge from the knowledge base to the current situation to decide on internal and external actions. The agent's current situation is represented by data structures representing the results of simulated sensors implemented in the interface and contextual information stored in the inference machine's internal memory. The inference machine must select and execute the knowledge relevant to the current situation...The inference machine cycles through a perceive, think, act loop, which is called the decision cycle.

1. Perceive: Accept sensor information from the game
2. Think: Select and execute relevant knowledge
3. Act: Execute actions in the game

[Ibid]

In order for an agent to solve a problem it must be able to infer new knowledge from existing knowledge and this is performed by the inference machine. We can identify the inference machine as being the part of the agent which performs the task of reasoning. It is the job of the inference machine to search the knowledge base and choose an appropriate action with regards to the context of the current game situation and as such must be reactive and able to quickly respond to information from the game. This is done by comparing all possible actions and attempting to choose the best one, which may vary from a random choice to implementing complicated planning methods.

3.1.2 Interface

The interface extracts all the necessary knowledge from the environment and encodes it in a format required by the inference machine. Each new game requires a new interface because the details of the interaction and the content of knowledge vary from game to game.

[Ibid]

The interface allows the agent to interact with the game world by passing knowledge to the inference machine. It is vital that an agent has a well designed interface in order that the inference machine has access to the appropriate knowledge in order that it may correctly make decisions within the context of the game.

3.1.3 Knowledge base

Informally, a knowledge base is a set of sentences. (Here “sentence” is used as a technical term. It is related but is not identical to the sentences of English and other natural languages.) Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.

[Russel & Norvig, 2003: 195]

The knowledge base contains all the possible actions which may be performed by the agent. This may include internal actions whereby knowledge is added or removed from

the knowledge base and external actions which involve making changes to the game world. Both the quality and quantity of the knowledge in the knowledge base will have a critical affect on the performance of the agent. If the agent is to provide solutions to complex problems it will need a large knowledge base to be able to react to different situations. The quality of the sentences will determine how realistic the actions of the agent are and a common cause of unrealistic behaviour in agents is 'unrealistic or missing knowledge in the knowledge base' (Laird & van Lent, 1999).

3.1.4 Requirements of an agent

Once the inference machine, interface and knowledge base are in place, Laird and van Lent (Ibid) propose five qualities which agents supported by the engine should display. Agents must be *reactive* in order to respond to changes in the game world and exist as part of an interactive environment. Agents must be able to choose *appropriate* actions according to the context of the situation and also must behave in a *realistic* manner (given that the purpose of the agent is to imitate human behaviour). Agents must be *flexible* and function in a wide variety of situations in order to provide the player with as much freedom as possible. Finally, agents must be *easy to program* and use a knowledge representation language which relates well to human conceptual understanding and is not abstract to programmer's thought process.

3.2 AI for Game Audio

In previous sections, the challenges of providing interactive audio for games were discussed and identified as the main barrier against applying many of the advanced film audio techniques was the lack of creative influence at run-time. AI has several factors which make it a good candidate for meeting these challenges. These are:

- One of the primary goals of AI research is to model or imitate human behaviour. By designing a program at run-time which can react to the changing situation of the game and make similar choices to a human composer we remove the most significant challenge in interactive audio.
- AI is most efficient when used for highly specialised tasks (Cawsey, 1998). By designing agents which deal with a limited number of specific tasks in music and/or sound design, the efficiency of the agents can be increased.
- AI has a long history of academic development and is a popular subject of research. When designing AI for audio, we do not have to start from scratch and can make use of existing research, including engines and architectures.
- There is a strong precedent for using AI in games which means many of the challenges of integrating AI into game programs have already been faced and overcome and we can focus on those AI techniques which already have proven to be successful.

AI provides a platform for interactive audio development which is both well established in the games industry and would be capable of handling a variety of advanced audio tasks.

3.2.1 Tasks for Audio AI

From previous chapters there are five main tasks that a game audio agent should perform.

- The agent must imitate human behaviour, specifically that it must make appropriate decisions which are perceived as both creative and appropriate.
- The agent must be able to perform transitions between segments of music in reaction to in game events. This includes selecting an appropriate segment of

music to change to and performing an appropriate transition. Planning and/or anticipation techniques may be used to provide a capacity for gradual transition and preserve continuity.

- The agent should be able to make use of asynchronous audio. This involves choosing an appropriate moment when asynchronous music and/or sound effects would enhance the emotion of the score and/or soundtrack. By default this task also requires the agent to determine when it is appropriate to choose synchronised audio or even silence.
- The agent should provide structure to the music and sound both in the long and short term. This involves developing the music over time to follow the development of the plot and careful choice of segments to provide self-similarity and repetition. Pre-defined structures such as sonata form may be implemented, or probability based algorithms such as those using 1/f noise (Dodge & Jerse, 1997).
- The agent should be programmed in a knowledge representation language which is compatible with the concept of sound being associated with, or an attribute of objects or even other associations or attributes. This facilitates the use of sound's associative nature as a creative tool and provides a common point of reference for programmer and artist.

We have identified the key tasks that AI may be used for in game audio; in order to develop solutions for these tasks, we must consider the possible AI techniques available.

3.3 AI Techniques in Games

Given that there is a rich heritage of AI involvement in games, it is prudent to consider different AI techniques with regards to how they have previously been used

in games and what benefits they might have for a game audio agent. Because of the wide range of techniques within the discipline of AI only a selection of the most relevant are considered.

3.3.1 Expert Systems

An expert system is a program dedicated to solving problems and giving advice within a specialised area of knowledge – such as medical diagnosis, automobile design, or geological prospecting. A good system can rival the performance of a human specialist in the area. Expert systems are AI's 'El Dollarado', and droves of new software companies have sprung up to exploit this lucrative breakthrough in programming technology.

[Copeland, 1993: 30]

Expert systems provide specialised agents which are highly focussed. The great benefit of an expert system is its ability to consistently provide the best solution with a high degree of reliability. This is ideal for purposes such as medical diagnosis and programs such as Internist (Cawsey, 1998) have enjoyed widespread success. Unfortunately, expert systems can require extensive development (usually including thorough interviewing of a human expert) and are not good at general problem solving so are suited to games with a strictly defined set of rules. Deep Blue, the chess playing supercomputer which famously beat grandmaster Gary Kasparov is a good example of a game based expert system. Because of their inflexibility, however, expert systems are not suited towards modelling creative behaviour such as composition.

3.3.2 Case-based reason

Case based reasoning techniques attempt to analyze a set of inputs by comparing them to a database of known, possibly historical, sets of inputs and the most advisable outputs in those situations. The approach was inspired by the human tendency to apprehend novel situations by comparing them to the most similar situations one has experienced in the past.

[Rabin, 2002: 5]

Because of its basis in human reasoning and historical human actions, case-based reasoning techniques can provide realistic human-like reactions. Despite this, case based reasoning is not frequently found in games as it relies on an extremely large knowledge base to deal with complex situations and is therefore inappropriate for game audio applications.

3.3.3 Finite-state systems

Finite-state machines are simple, rule-based systems in which a finite number of “states” are connected in a directed graph by “transitions” between states. The finite-state machine occupies exactly one state at a time.

[Rabin, 2002: 6]

Conceptualising a problem in terms of states can be useful in terms of games design, where the “state” is a representation of the game world and can encompass many factors including environment, emotion of the character and plot. The action of making transitions is of primary concern in developing game audio AI which means that a finite-state system may be particularly applicable to real-time composition. Some contemporary methods of non-linear composition already make use of state-based systems (Morton, 2005b).

3.3.4 Production systems

Production systems are comprised of a database of rules. Each rule consists of an arbitrarily complex conditional statement, plus some number of actions that should be performed if the conditional statement is satisfied. Production rule systems are essentially lists of “if-then” statements, with various conflict resolution mechanisms available in the event that more than one rule is satisfied simultaneously.

[Ibid]

Production systems have benefited from great popularity within games design. The basic component of “if-then” statements is one of the most fundamental functions in the C language which is commonly used for game programming, facilitating the integration of

production system AI with other game code. Conflict resolution strategies mean it is possible to have a number of outcomes for any given situation; increasing both flexibility and variety, both of which are valuable assets when simulating creativity within a game audio agent.

3.3.5 First order (predicate) logic

The most important knowledge representation language is arguably predicate logic (or, strictly, first-order predicate logic). Predicate logic allows us to represent fairly complex facts about the world, and to derive new facts about the world, and to derive new facts in a way that, if the initial facts were true, then so are the conclusions. It is a well understood formal language, with well-defined syntax, semantics and rules of inference.

[Cawsey, 1998: 20]

First order logic is one of the most powerful methods of knowledge representation. A system using first order logic has the ability to infer accurate, new knowledge about the world meaning that when faced with new situations, it will often be able to find a solution by creating new knowledge. First order logic is also based around an object-property system which facilitates the expression of sound as an attribute. Unfortunately, first order logic is generally too computationally expensive to feasibly be applied to computer game AI with modern technology.

3.3.6 Semantic networks

Semantic networks were originally developed in the early 1960's to represent the meaning of English words. They have since been used more widely for representing knowledge. In a semantic network knowledge is represented as a graph, where the nodes in the graph represent concepts, and the links represent relations between concepts.

[Cawsey, 1998: 14]

One of the advantages of using semantic networks to represent knowledge is that, having been designed as a model for the English language, their nature is easy to comprehend even for non-programmers or AI specialist and supports the notion of

representing object-attribute relationships. In addition, object oriented programming languages, such as C++ are based on the concept of semantic networks, meaning that most games programmers will find the concept familiar. Figure 3.1 shows an example of a semantic network describing animals.

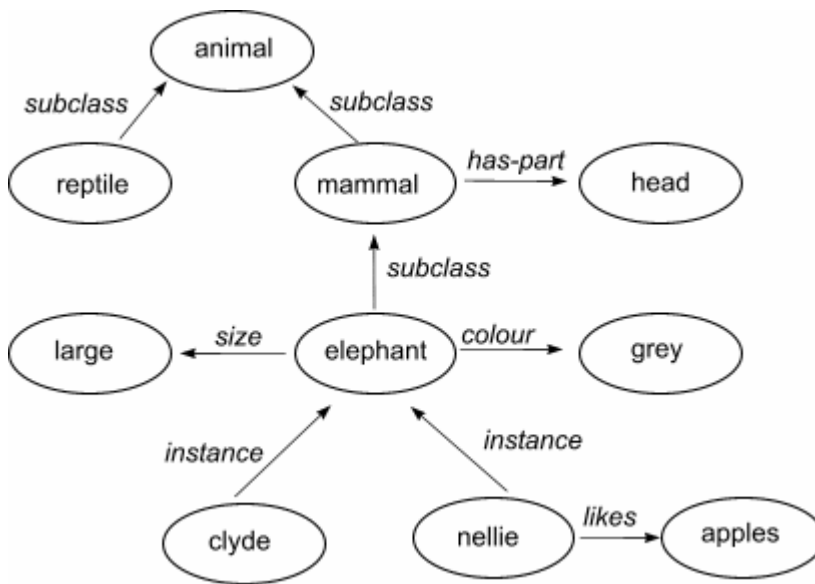


Figure 3.1 Semantic network (After Cawsey, 1998)

In this network an elephant is described in terms of being a sub-class of mammals; from this we can (correctly) infer that an elephant has a head, since it is a mammal and mammals have heads. Conversely, although elephants are grey, not all mammals are since this is an attribute of elephants and not the mammal class in general.

3.3.7 Genetic Algorithms

Evolutionary algorithms are a collection of solutions based on the theory of evolution. As such, genetic algorithms are a popular process that can optimize parameters in multi-dimensional space. Genetic algorithms use a simulation of the problem to determine the fittest solutions among the population.

Improvements over time are encouraged by only letting the fittest individuals survive and contribute to the gene pool.

[Champandard, 2004: 421]

Genetic algorithms have been employed in games with a good degree of success. The ability of the algorithms to self program and optimise makes them useful for agents which need to adapt to new environments and situations. Quake 3 made use of genetic algorithms in the weapon selection mechanisms of artificial opponents, allowing them to choose the appropriate weapon in any situation. Although the ability to adapt is beneficial to a game audio agent, genetic algorithms are not suited to imitating human creativity as they are focussed on optimisation.

3.3.8 Neural networks

The mechanics of biological neural networks are not fully understood; however, the basic architecture of these biological constructs and their functionality can be modelled to some degree. Through the use of mathematics, electronic circuitry, or computer algorithms, the interaction between neurons can be simulated and has achieved overwhelming success in the field of artificial intelligence. Many characteristics once associated only with biological neural networks such as learning, generalisation, interpretation, and prediction can now be duplicated.

[Rogers, 1997: 2]

Neural networks are one of the most powerful AI techniques and are abstract from all other techniques in that they do not attempt to model a reasoning process, but instead model the biological process which humans use for reasoning. The main argument against using a neural network is that there is no real understanding about the internal semantics in any but the simplest of networks. In order to use a neural network, it must be trained so that with a given state across its inputs it provides the desired state across its outputs. In a simple case this is not particularly difficult, however for more complicated situations (such as may be expected in a game world), a large amount of training data is needed. Due its ability to learn, it is theoretically possible that a network would be trained to imitate human behaviour such as compositional technique, as attempted by Moser

(1994), particularly for short sections of music such as transitions. Although there is a lot of potential for the use of neural networks within game audio agents, the fact that there is a lack of understanding about their operation and a lack of an established presence within game design means that it is worth developing game audio agents with proven (and simpler) techniques before attempting to do so with neural networks.

3.3.9 Suitable techniques

From this brief assessment of AI techniques, there are a number which are particularly suitable for a game audio agent. Finite-state systems, production systems and semantic networks all provide simple but effective AI design that is well suited for use in games. These techniques allow agents to be flexible and imitate human behaviour and have features which translate well into the C/C++ programming domain as well as a proven history in game design.

One AI architecture which includes all three techniques in its design and is freely available for research and development is Soar.

3.4 Soar architecture

Originally designed in an effort to provide an architecture which supported development of a wide range of AI applications, Soar's history includes applications within both games and military exercises in addition to more traditional AI research such as problem solving:

Soar has been used for research in AI, including problem solving, planning and learning. Soar has also been used as the basis for detailed modelling of human behaviour... The application of this technology to computer games appear obvious. Current first-person perspective competitive games, such as Doom, Quake or Decent, have relatively simple AI opponents and depend on numbers

and fire power to make the game interesting. We wish to explore games where the AI opponent is on par with humans, so that each level is more like a deathmatch against one or two opponents, than a slugfest against many. We also envision high-fidelity AI systems that can substitute for real players in long-term role playing games, where continual human play is not possible.

[Laird & Jones, 1998]

Soar has many features which make it an ideal platform for developing game audio AI.

The existence of previous research into applications of Soar for computer games is a significant benefit as it means the focus can remain on developing the audio agents rather than building an engine from scratch. Human behaviour modelling is a key concept in terms of simulating compositional creativity and Soar has been proven to be adept at this while its design as a non-specific AI engine allows for a great deal of freedom in developing a game audio agent. Current Soar-games research is investigating the use of Soar agents as a method of plot management which supports the concept of a game audio agent providing plot-based structure to the music. In operation, Soar is a production based system which additionally uses finite-state mechanics and a semantic network style knowledge representation language; combining the advantages of each approach. Productions may be loaded to an agent at any time, meaning that there is a great scope for reusability of code and for updating or customising the agent.

3.4.1 Basic Soar operation

The design of Soar is based on the hypothesis that all goal oriented behaviour can be cast as the selection and application of operators to a state. A state is a representation of the current problem solving situation; an operator transforms the state (makes changes to the representation); and a goal is the desired outcome of the problem solving activity.

[Laird & Congdon, 2005: 5]

To support the operator based nature of Soar whilst retaining flexibility, the architecture has been designed around the following two fundamental concepts.

1. In order to allow Soar to be applicable in a wide range of AI tasks and to simplify the development of agents, the architecture provides four distinct mechanisms; productions, working memory, automatic sub-goaling and chunking. Productions form the long term memory of a Soar agent and consist of a collection of conditional (if-then) statements which propose and apply operators (i.e. transitions of state). Working memory represents Soar's short term knowledge in the form of a network of working memory elements (similar to a semantic network) which includes all knowledge about the current state; both external (from the game) and internal (results of productions applying an operator). Automatic sub-goaling is the mechanism used by Soar to generate goals allowing agents to solve complex problems and state transitions. One of the major advantage an AI system such as Soar has over hard-coding behaviour in C/C++ (as is common in the games industry (Laird & Lent, 2000)) is that a small number of generalised rules can replace many explicitly programmed rules. Chunking allows Soar to learn by storing successful solutions to a problem as productions in the long term memory; however it is not always necessary to activate this feature.

2. Soar makes all decisions at run time with the relevant knowledge from production and working memory; scripted or pre-written sequences of actions are never used when programming Soar agents; instead sequences of actions can be executed using sub-goaling. This means that the agent is always reactive to the current situation, even when conducting complicated tasks that need to be broken down. For example, an agent performing run-time composition receives input which indicates a change of musical segment is necessary. The agent selects a new segment and begins to transition but during the transition the input indicates that the game situation has changed; requiring a different theme to be selected. Instead of attempting to complete the original transition, the agent will be able to respond to the new knowledge.

3.4.2 Soar functionality

All of Soar's long-term knowledge is organised around the functions of operator application and selection.

[Laird & Congdon, 2005: 6]

Programs for Soar are written as productions which form the long-term memory of Soar. Productions tell Soar how to change the state and achieve goals by performing one (or more) of the four operator based functions; proposal, comparison, application and elaboration. Of these four, proposal, comparison and application are core to the functionality of soar while elaboration is used to simplify programs by representing 'Knowledge of monotonic inferences that can be made about the state' (Ibid).

As previously mentioned, productions are conditional functions and are expressed in terms of their conditions and actions. If the conditions of a production are met by elements in the working memory they are said to *match*; the production will *fire* and perform its actions; usually a manipulation working memory elements (which include outputs to the game environment). An abstract example of a production for proposing an operator is:

Conditions: Thirsty
 Beer

Actions: Propose operator: Drink beer

In this case, if the working memory elements (WME's) Thirsty and Beer, the production will match and will propose the operator Drink Beer as being suitable for selection (which involves creating an operator-proposal WME). Proposing operators is the first step in performing any action and Soar can be conceptualised as running through a continuous process of proposing and selecting operators. If there is only one operator, Soar will select that operator. A state always has one operator and every decision cycle will attempt to replace that operator with a new one. Often, however, several productions will

simultaneously match and Soar will have to choose between several possible operators. This is achieved by productions which perform operator comparison. Consider the following situation in which the working memory contains the elements Thirsty, Beer and Water, and the long-term memory contains the following productions:

Production 1
Conditions: Thirsty
 Beer

Actions: Propose operator: Drink beer

Production 2
Conditions: Thirsty
 Water

Actions: Propose operator: Drink water

In order for Soar to choose between operators, working memory must contain a *preference* which compares the operators. In the following example, the production tests that there is a state, which is equivalent to testing for 'I exist' and is always true.

Production 3
Conditions: I exist

Actions: Create preference: beer is better than water

In this example Soar would select the operator for drinking beer. Creating preferences is an important function and will ultimately decide how appropriately Soar will act in different situations. For example, we may want to include the following production:

Production 4
Conditions: Hungover

Actions: Create preference: never drink beer

With the inclusion of this production, Soar will always choose to drink beer unless the WME Hungover exists, in which case it will choose water. Although beer is always better than water, it is never drunk when hungover.

An operator selection can be thought of as Soar making a decision to do something, in order for that decision to be carried out, it must be applied:

Production 5
Conditions: Operator: drink beer

Actions: Remove: Beer
 Remove: Thirsty

This is an example of Soar performing an action internally and is analogous to Soar thinking in its head. In a game situation, however, we are likely to want to have actions performed externally, in which case we would use the following production instead:

Production 5
Conditions: Operator: drink beer

Actions: Add to output: drink beer

In this case the WME's that represent being thirsty and beer are not removed until new knowledge is received from the input informing Soar that the beer is has been drunk.

This is most effectively performed using an elaboration of the state:

Production 6
Conditions: Input-Beer: drunk

Actions: Remove: Beer
 Remove: Thirsty

An elaboration of the state performs no operator functions but makes changes to the state as a result of testing WME's in the state (commonly the input from the game). Elaborations are useful in simplifying productions which apply operators and improve the flexibility of the agent (Nuxoll & Laird, 2003).

Soar operates in five-stage cycles, shown in Figure 3.2

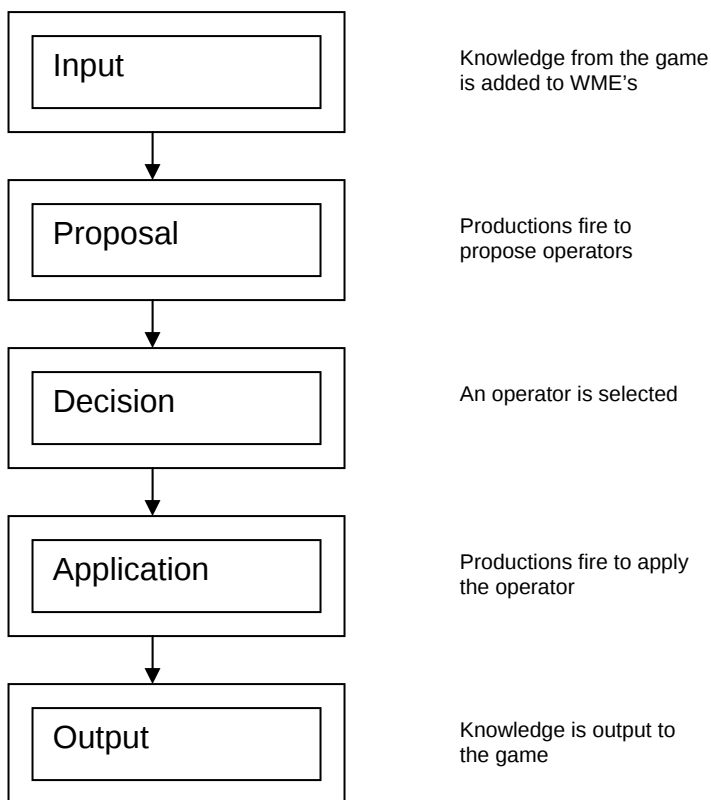


Figure 3.2 Soar operation cycle

Soar continually moves through the cycle, attempting to select a new operator each decision phase from the preferences created during proposal. In the proposal and application phases, productions fire in simulated parallel, that is all the productions that match fire, then any new matches from the result of productions firing and so on, until there are no further matches.

3.4.3 WME support

All productions perform their individual tasks of proposal, comparison, application and elaboration by adding elements to the working memory; however it is important to differentiate between those elements which should be persistent and those which should not. Conceptually, WME's created as a result of an operator application represent physical changes to the state and must remain even when the production which created them no longer matches (i.e. when a new operator is selected) and are said to have operator support or o-support. Once the beer has been drunk, it stays drunk. Conversely preferences and comparisons can only be supported while the productions which created them match and this is known as instantiation support or i-support. If you stop being thirsty, you will no longer want a drink. Elaborations also have i-support which allows them to dynamically respond to changes in the state.

3.4.4 Preferences

There are nine types of preference used by Soar to compare operators.

- Acceptable (+) indicates that the operator may be a candidate for selection.
Operators must be acceptable (or required) to be considered for selection
- Rejected (-) indicates that an operator is not a candidate for selection
- Indifferent (=) indicates that an operator is neither better nor worse and may be used globally or to compare specific operators. If several indifferent operators tie as the preferred choice, one is selected randomly.
- Numeric-indifference (= number) is the same as indifferent except that a number is used to bias the random selection.

- Best (>) indicates that an operator is better than any other choice unless they are better than the operator or required.
- Worst (<) indicates that an operator is worse than any other choice unless they are worse than or rejected.
- Better than (> operator) indicates that an operator is better than another.
- Worse than (< operator) indicates that an operator is worse than another.
- Required (!) indicates that an operator must be selected regardless of any other preference.
- Prohibit (~) indicates that an operator cannot be selected regardless of any other preference.

Preferences are used by Soar to choose the best operator for a given situation, and therefore the best way to advance the state towards the goal. Using preference-based comparisons for decision making is potentially very useful in audio agent design as it is analogous to human reasoning in tasks such as composition, where the composer has to choose an appropriate musical theme, style or transition or even when to apply asynchronous sound techniques.

3.4.5 Impasses

If Soar is unable to select an operator for any reason, an impasse occurs and Soar creates a sub-state with the goal of resolving the impasse. There are two basic types of impasse which may occur: preference and no-change. Preference impasses occur when the preferences do not suggest one operator as the most preferable which may be due to a tie; conflicts between operators being better or worse than each other or an operator having both require and prohibit preferences. No-change impasses occur if a new operator is not selected at the beginning of a cycle due to there being no acceptable

operators for the current state (state no-change), a new operator is selected and no productions match to fire in the application stage, or the same operator is the most preferable in consecutive decision phases.

The creation of sub-states and sub-goals plays an important role in both Soar's problem solving and learning mechanisms. It is possible that a sub-state created to solve an impasse will also result in an impasse, in which case a new sub-state is created to resolve the new impasse. Figure 3.3 shows an impasse resolution which is two sub-states deep.

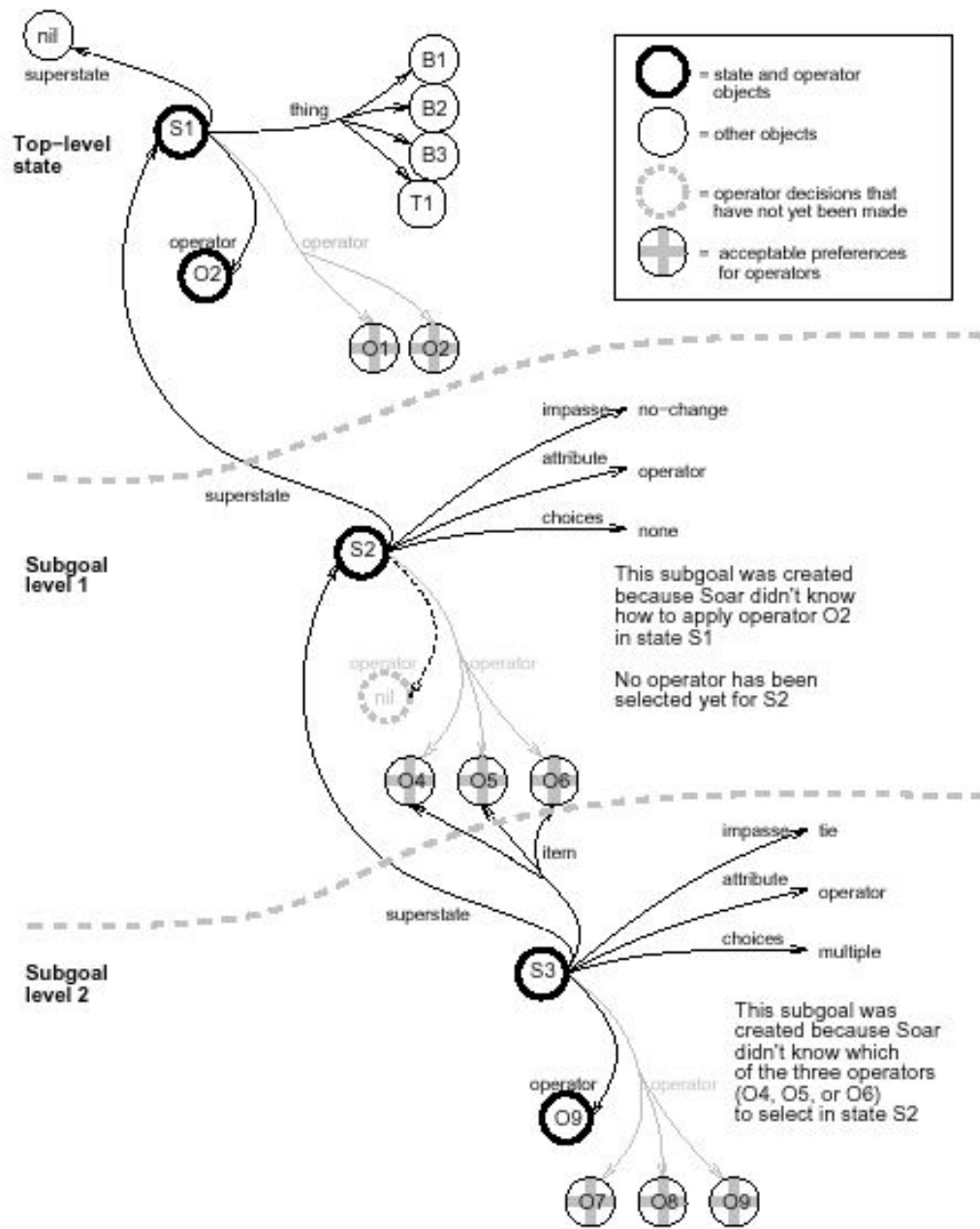


Figure 3.3 Soar sub-goaling [Laird & Congdon, 2005]

Soar will continue to sub-goal until either the impasse is resolved or new inputs cause a viable operator to be proposed. If learning is enabled and Soar resolves an impasse through sub-goaling, a *chunk* is created which is a new production that summarizes the

process that led to resolution of the impasse. If a similar impasse arises again, the chunk will tell Soar how to solve it.

3.4.6 Syntax

Knowledge in Soar is divided into WME's and preferences (stored in working memory), and productions (stored in production memory). Each type of knowledge has its own syntax, used when programming Soar. WME's and preferences are never hard-coded into Soar, they are always the result of production firings.

A WME is a list consisting of three symbols: an identifier, an attribute, and a value, where the entire WME is enclosed in parentheses and the attribute is preceded by an up-arrow (^). A template for a working memory element is:

(identifier ^attribute value)

The identifier is an internal symbol, generated by the Soar architecture as it runs. The attribute and value can be either identifiers or constants; if they are identifiers, there are other working memory elements that have that identifier in their first position. As the previous sentences demonstrate, identifier is used to refer both to the first position of a working memory element, as well as to the symbols that occupy that position.

[Laird & Congon, 2005: 33]

WME's represent knowledge about the state in a form similar to a semantic net. Objects within the state are treated as attributes of augmentations of the state and augmentations may in turn have one or more augmentations. A WME with no augmentations is known as a value. Soar makes no distinction between physical objects and other attributes such as red, above, happy etc., so a series of WME's representing a black cat in a basket could be expressed as:

```
(state <s>)  
(^basket <bas>)  
(<bas> ^in <contents>)  
(<contents> ^cat black)
```

The state is always the top level in a WME chain and WME which is not linked to the state is removed from working memory. Comments enclosed by <> are variables used to match identifiers. Identifiers are never entered manually when programming Soar since they are generated at run-time but may be represented by variables. Representing the state in this way is, coincidentally, a benefit in terms of sound design since it facilitates the conceptualization of sound as an association or attribute.

A collection of WME's that share an identifier is known as an object, for example:

```
(state <s> ^animal <c>)  
(<c> ^color black)  
(<c> ^species cat)  
(<c> ^personality lazy)
```

represents a lazy black cat. The augmentations must refer to the same animal as they all have the same identifier (i.e. the value stored in variable <c>).

Preferences are created by production firings and express the relative or absolute merits for selecting an operator for a state. When preferences express an absolute rating, they are identifier-attribute-value-preference quadruples; when preferences express relative ratings, they are identifier-attribute-value-preference-value quintuples. For example,

```
(S1 ^operator O3 +)
```

is a preference that asserts that operator O3 is an acceptable operator for state S1, while

```
(S1 ^operator O3 > O4)
```

is a preference that asserts that operator O3 is a better choice for the operator of state S1 than operator O4.

[Laird & Congon, 2005: 38]

Preferences serve a highly specialized role within the Soar architecture and as such have a very specific and brief syntax to represent its knowledge. They are added to working memory (or rather preference memory which is a sub-section of working memory) as a result of productions firing.

Productions control the operation of Soar by controlling the selection and application of operators. A Soar programmer writes a collection of productions which are loaded into the production memory of a Soar agent at run-time. Productions may be loaded into an agent at any point in operation or even typed in manually when using the Soar debugger. Laird & Congdon (2005: 39) give the basic structure of a production as:

```
sp {production-name
Documentation string
:type
CONDITIONS
-->
ACTIONS
}
```

The documentation string and type are optional components of a production and for the sake of simplicity shall not be used in further productions. Here is an example of a production used in an agent built to tackle the classic AI water jug problem,

```
#water-jug*propose*fill
#If the task is water-jug and
#there is a jug that is not full,
#then propose filling that jug.

sp {Water-jug*propose*fill
    (state <s> ^name water-jug
        ^jug.empty > 0)
→
    (<s> ^operator <o> + =)
    (<o> ^name fill
        ^fill-jug <j>)
}
```

This production is used to propose an operator to fill a jug if it is not full.

```
#water-jug*propose*fill
#If the task is water-jug and
#there is a jug that is not full,
#then propose filling that jug.
```

Although not necessary, it is good practice to begin all productions with a comment which includes its name and explains, in English, the function of the production.

```
sp {Water-jug*propose*fill
```

The name of the production is declared. A production's name should include, at least, the name of the problem (or agent function in a game), the type of operator function and the name of the operator to aid debugging.

```
    (state <s> ^name water-jug
      ^jug.empty > 0)
→
```

The conditions of all productions begin with a declaration of the state. Here the conditions test that there is an augmentation of the state called `name` with a value of `water-jug` and an augmentation of the state called `jug.empty` which has a value greater than 0. The production is testing that goal is the water-jug problem (given by the name of the state) and that there exists a jug within the state that is not full (literally: more than not-empty)

An arrow (- - >) signifies that the conditions have finished and the actions are about to be stated.

```
    (<s> ^operator <o> + =)
      (<o> ^name fill
        ^fill-jug <j>)
    }
```

The actions of this production create acceptable and globally indifferent preferences for an operator with an augmentation `name` that has a value of `fill` and an augmentation `fill-jug` that has an identifier as its value. Note it is important to link the operator preference to the state with the variable `<s>` which was declared as the identifier-value of the state in the first line of the conditions. The augmentations of the preference identify the name of the operator and the jug it is proposing to fill.

3.4.7 Development software

There are two main pieces of development software which are useful to the Soar programmer and are distributed as part of the Soar suite; Visual Soar and the Soar Debugger.

Visual Soar is a programming environment for Soar, written in Java and is shown in Figure 3.4.

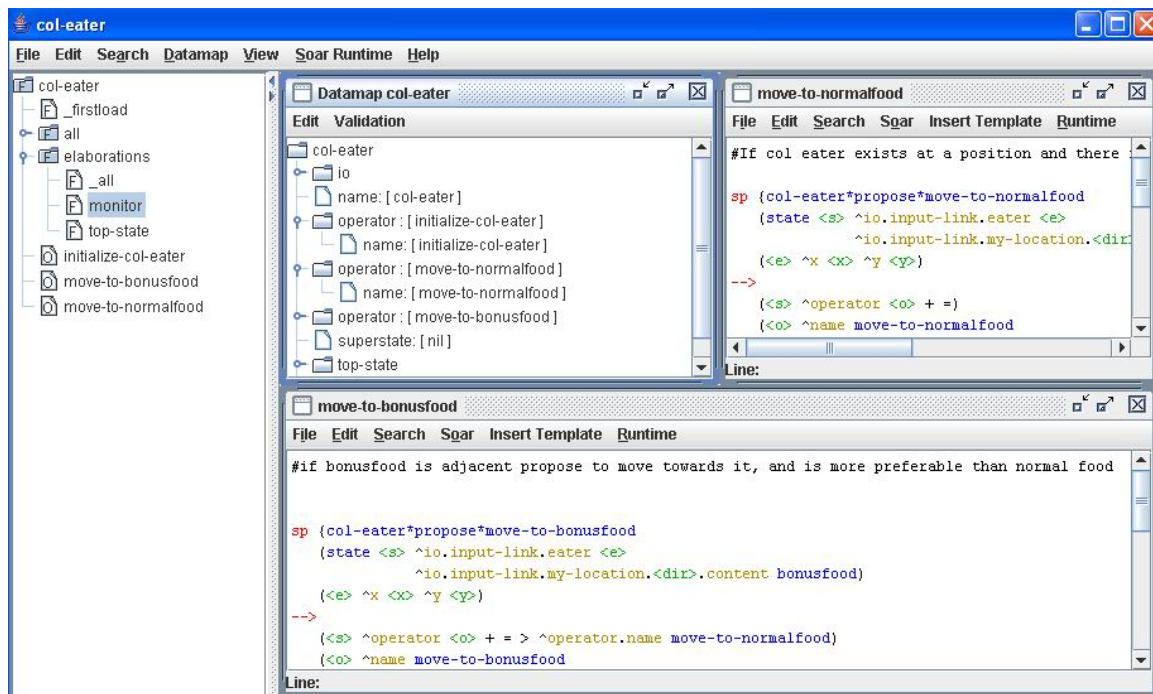


Figure 3.4 Visual Soar programming environment.

Production code is colour coded and automatically formatted to aid visual inspection with identification and distinction between WME's, variables and values made clear. The windows at the bottom and right of the screen display production rules for the operators move-to-bonusfood and move-to-normalfood. On the left hand side the project is

presented in terms of the operator hierarchy which is invaluable for larger scale projects which make use of subgoalting. Productions are organized by operator and a datamap window (top-center in Figure 3.4) can be opened which provides a representation of working memory and can be used to automatically check for productions that match specific elements or even do a project wide scan to check productions against the datamap and syntax errors.

The Soar Debugger, like Visual Soar, is written in Java and provides an indispensable tool for programming soar. Figure 3.5 shows the Soar Debugger interface.

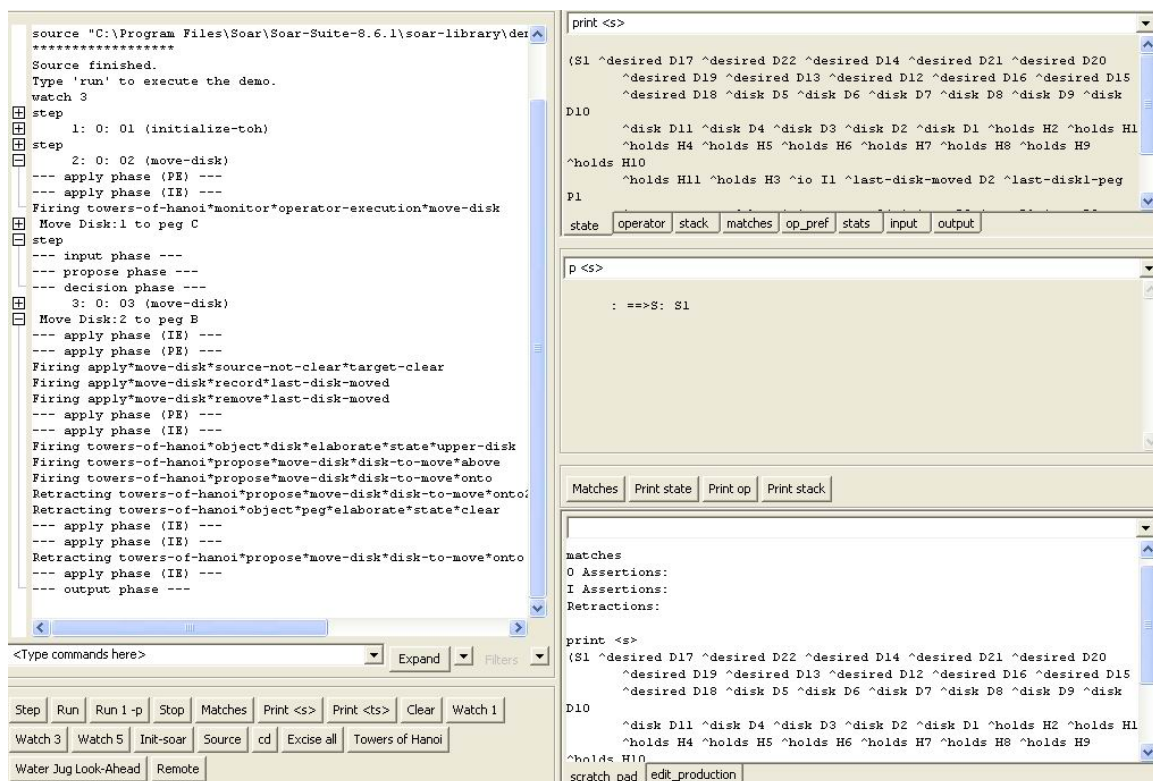


Figure 3.5 Soar Debugger interface

In order to check that a Soar program is functioning, it can be loaded into the Soar Debugger which features a number of powerful features. Alternatively, the debugger can be used remotely on a Soar kernel which is running in a host program (e.g. when being

implemented within a game). The left window in the interface shows a list of operations in Soar; the level of detail displayed can be varied between only showing operator selection and showing every production that fires. The view is explodable, so when using a high detail view it is possible to minimize and expand the individual operations. It is possible to run Soar at normal speed or to move step by step through the operation to inspect, in detail, the actions of the agent. The windows on the right of the interface provide information on the current state, operator, matches and performance in addition to an entry field where new productions can be written and loaded into Soar while debugging.

3.5 Soar research on computer games

Research into the application and development of Soar has included several high profile computer game projects including building adversarial agents in Quake II and Descent 3, and, more recently, developing agents to manage plot in interactive drama. AI research using computer games is mutually beneficial both commercially and academically. As computer games become more advanced, the demand is becoming greater for highly developed AI to provide more realistic NPC's and a more immersive experience; at the same time games provide a detailed simulated world which allows for academic research to concentrate on developing better AI programs and not be concerned with the limitations of robotics and other factors commonly found in traditional AI research methods (Laird & van Lent, 2000).

3.5.1 Soar Quakebot

Soar Quakebot is an agent designed to control a NPC opponent for a human player in the first-person shooter game Quake II. Although the high-level purpose of the Quakebot

is largely at odds with using AI for game audio, there are many techniques related specifically to modeling human behavior, interfacing with the game environment and anticipation using Soar.

A common shortfall of AI in computer games is that is either not very reactive, or is not context specific. AI agents which rely upon a large number of low-level conditional statements (such as if-then in C) are typically very reactive but have no way of representing context as they lack any internal memory. Although the agents can react believably at any one instance, their behavior is limited to acting in the moment and is inconsistent in the long term; musically this results in a lack of structure and development. Another traditional method of behavior modeling is to use scripts: a series of pre-programmed actions which are triggered by specific in-game events. Because the actions are pre-programmed (and therefore effectively linear) the agent has a very strong sense of context and can act consistently; reactivity is sacrificed, however, and if the game-state changes during the script all sense of believability is lost.

The ability to react to the state of the game while providing context in terms of structure and thematic development is vital for an advanced game audio agent. Quakebot tackled this challenge by exploiting the sub-goaling function of soar to create a hierarchy of states which provides a context for the action being performed at any given point in time (Laird & van Lent, 1998). Figure 3.6 shows a sample operator hierarchy for a Quake style action game.

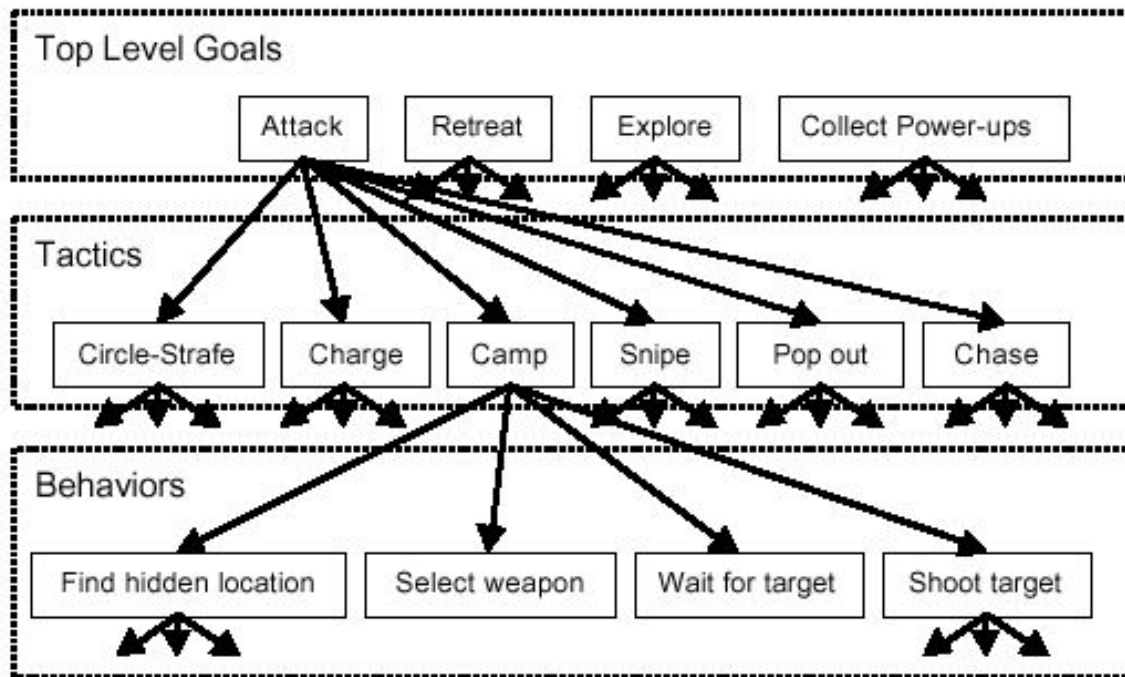


Figure 3.6 Sample operator hierarchy for Quakebot style agent [Laird & van Lent. 1998]

In a given hierarchy, the same low level operator may appear several times, but each time in a different context; in the example shown in Figure 3.6, the *Shoot target* operator is likely to be repeated in all of the *Attack* sub-operators. An operator hierarchy such as this also has the advantage of being conceptually similar to the way a human might approach a task by breaking it down into a number of smaller tasks. A game audio agent must be able to imitate a human composer and/or sound designer; by using a reasoning model which is analogous to human problem solving, we are not only likely to develop an agent which performs with a greater degree of realism, but which is also easier to program.

A core component of a game agent is its interface with the game world. In the case of Quakebot, the interface was designed to provide the agent with the same information that would be available to the player (Ibid). A common approach with basic

implementations of basic AI methods in games is to allow the AI agent to cheat by having access to extra information; without the extra information the AI is easily defeated by a human player. While cheating may make the game challenging, it is often very obvious to the human player and is therefore counterproductive to a sense of immersion. A game AI agent is not acting as an actual character within the simulated world of the game and does not need to be restricted information available to the player. In many cases it may be advantageous for the agent to know of possible events ahead of the player so that it may alter the music and sound design accordingly; for example to create a feeling of suspense before facing a particularly dangerous monster. The lesson we must learn from the Quakebot interface, however, is that the player must not perceive that the agent is doing this. From the player's perspective, the music and sound are a part of the environment expressing the emotion of the game world and should be seen to be as such. When considering the interface for a game audio interface, we should not only consider basic inputs such as the player's location and 'mode' (e.g. battle mode, rest-mode, exploring-mode etc.), but try and capture the properties of the world which the audio expressing (such as the emotion of the game-character(s)). Rather than labeling a certain location or area as having a certain mood or emotion, the reasons for that mood or emotion should be available to the agent for inspection. An in depth discussion of how this might be achieved would, necessarily be highly detailed and is beyond the scope of this project.

Anticipation and planning can be great benefits for a game audio agent. By anticipating events in the game world, an agent can use this knowledge to create a feeling of suspense (common in horror films) or support emotional transitions in other ways. When used to support the plot, an audio agent with anticipation can help to promote certain actions by the player; either by using music to influence their emotions or sound effects to give gameplay cues; for example if the plot requires that a player inspects a

mysterious statue, as the player approaches nearby, the agent might transition the music subtly to reflect a sense of mystery and trigger a sound effect from the direction of the statue. Quakebot performs anticipation of opponents by creating a model of what it thinks is the internal state of its opponent and considering what actions the opponent might take. An analogous approach could be taken by a game audio agent to predict the actions of the human player. Planning is performed similarly; the agent would run an internal simulation of the action it was about to take (for example a transition between two pieces of music) and find a suitable way to perform that action before making any changes to the state of the game.

3.5.2 Interactive Drama Architecture

Developing AI to control opponents for human players is of limited interest both academically and in terms of gameplay. Academically, hostile NPC's only model a human's ability to fight tactically while researchers would aspire to create agents capable of more complex behaviors. In terms of gameplay, there is a thin line between an opponent which provides an engaging challenge and an opponent which is almost unbeatable (few computer chess games would be successful if they used AI as powerful as Deep Blue!). In order provide a more involving game experience and have an opportunity to develop more advanced AI agents, current Soar research into games is focusing on creating agents for plot based games:

...we are currently working to develop non-violent plot-driven computer games where the AI characters have diverse and complex behavior that is driven by the interaction of their body with the environment, their goals, their knowledge of the world they inhabit, their own personal history, and their interactions with human players. This will lead to games where the human players are faced with challenges and obstacles that require meaningful interactions with the AI characters. We are building on one of the oldest genres of computer games, sometimes called interactive fiction or adventure games, which involve having the human player overcome obstacles and solve puzzles in pursuit of some goal.

[Laird, 2002]

This change in direction of research towards plot orientated AI supports the potential for game audio agents built on Soar architecture. Not only will there be significant crossover of technical developments, but also an opportunity to use plot-agents and audio-agents together. The Interactive Drama Architecture proposed by Brian Majerko is a good example of plot based research that can be integrated into audio agent development.

[Interactive Drama Architecture] attempts to create an experience driven by a pre-authored plot, without explicitly limiting the User's actions, some of which may work directly against the intent of the plot. In order to allow the User as much freedom as possible, while still maintaining the narrative, we introduce an omniscient agent, the story Director, which performs three main tasks in order to protect the narrative's integrity: plot verification and behavior monitoring to make sure the user's behavior hasn't harmed the future narrative and is consistent with timing constraints of the plot, predictive modeling of the User to detect such errors before they occur, and preemptive direction to modify the behavior of the characters and the world to encourage the User to move along the narrative path and avoid complete plot failure.

[Magerko & Laird, 2003]

We can see an immediate parallel between the goal of IDA and the need for audio agents to provide plot based structure to music and sound design. IDA proposes a method of following and promoting the plot which can directly be adopted in the design on audio agents. By following the course of the plot and modeling the behavior of the player, an audio agent would have a reference to the current and past plot in addition to having some knowledge of what might soon happen and would be able to develop the structure of the music in the context both the plot and the current situation. It is important for the agent to be able to dynamically understand the context of past, present and future since, in a world where a player has freedom of choice, a single plot point may be reached by a potentially large number of routes.

In IDA, scenes in a plot are represented as states, where a plot state contains a number of assertions about the game world which represent objects or actions that must be in place for the plot to progress; for example if the player's character possesses 'a

faded map' and speaks to 'one-eyed pirate' the player may be told the location of 'buried treasure'. It is important to note that the states do not represent a complete image of the game world but only a few important facts, so it is possible for a plot point to be satisfied in a number of situations. Figure 3.7 illustrates an example sequence of plot states.

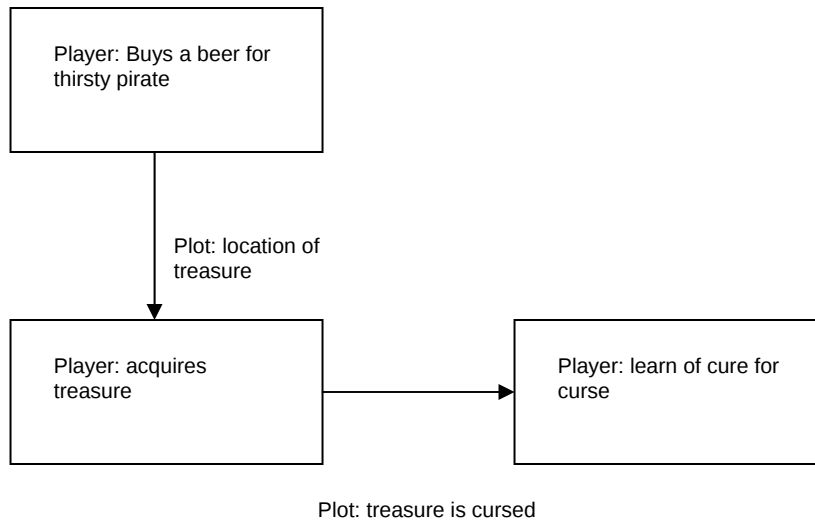


Figure 3.7 Example sequence of plot states

Here we see the plot states which contain the conditions for the plot to progress represented by boxes and the actions which connect them represented by arrows. Ideally, the player would remain oblivious to the presence of the plot states and would perceive the process as one continuous adventure. In the example, the second and third plot states are deliberately unspecific; there may be a number of ways for the player to acquire the treasure given that they know its location and likewise with finding a cure for the curse. Another method of promoting a sense of variety and free will or even to allow for several parallel plots to occur is to include plot states which branch of into two or more plot states; the player may have the option to progress through the drama as a good character or evil depending on their decisions.

As with IDA, audio agents would be a tool used to realize the artistic vision of a human as an interactive performance. A state based plot system is beneficial in a number of ways towards this goal. By presenting the composer with the plot in an abstract, state-based form he will be able to write a score which takes into account the progression and development of the plot in the large scale. Composition on this scale is likely to involve themes and leitmotifs that are linked to the plot and overall development of the score (e.g. the plot may get progressively dark and depressing throughout the game). A sub-plot version may also be used, in conjunction with the overall plot scheme, to represent music in the short term in a similar method to current interactive game composition; the composer writes a collection of musical segments providing a number of variations on generic states that the player may experience such as love, hate, danger and suspense. The states which express the short-term experiences of the play can be generated using a sound map of bi-polar plot extremes as discussed in Chapter 2 section 2.2.6.

Since the Soar architecture has its foundations in state-based problem solving, developing an agent which makes use of the plot states is simplified when writing a Soar program.

3.6 Programming Soar

Soar is programmed by writing the productions that propose, prefer and apply operators during Soar's operation cycle. To provide an understanding of Soar programming methods, we shall discuss the code for agents built to solve a simple problem, known as 'Water Jugs' (Laird, 2005) and to control the actions of a character in a simple game environment called 'Eaters' (Ibid). Example productions illustrating how Soar may be programmed for game audio are presented along with a C++ program to demonstrate the Soar interface.

3.6.1 Water Jugs

In the classic Water Jug problem there are two jugs, one which can hold 5 litres and one which can hold 3 litres. There is a tap which can be used to fill the jugs and a sink so that a jug may be emptied. The goal is to fill the three litre jug so that it has exactly one gallon of water by filling, emptying and pouring water between the jugs. A simple way to solve this problem in Soar is to create a representation of the state in working memory and then propose operators to fill, empty and pour water until the goal is achieved.

The first step in solving the problem is to initialize the state by creating a representation of the jugs and their properties in working memory:

```
#If no task is selected, propose the water jug task

sp {propose*initialize-Water-jug
    (state <s> ^superstate nil
      -^name)
-->
    (<s> ^operator <o> +)
    (<o> ^name initialize-Water-jug)
}
```

First an operator is proposed to initialize the state and is usually the first action taken in any Soar program. The production tests that the current state is the top-state and that there is no name for the state and therefore no current goal.

```
#if the water jug task is selected:
# initialize the water-jug problem

sp {apply*initialize-Water-jug
    (state <s> ^operator.name initialize-Water-jug)
-->
    (<s> ^name water-jug
      ^jug <j1>
      ^jug <j2>)
    (<j1> ^volume 5
```

```

        ^contents 0)
(<j2> ^volume 3
    ^contents 0)
}

```

Since, at the beginning of the program, the only operator proposed will be initializing the state, it will always be selected. The actions of this operator create augmentations on the state to represent the two jugs; each of which has augmentations to represent volume and contents. When proposing operators for this problem, we may want to consider the problem in terms of how empty a jug is as well of how full. A jug's emptiness is simply its volume minus its contents, which we calculate with the following production:

```

# If the state is named water-jug and a
# jug can hold volume v and currently has
# contents c, then add that it has v - c
# available (empty) space.

sp {Water-jug*elaborate*empty
    (state <s> ^name water-jug
        ^jug <j>)
    (<j> ^volume <v>
        ^contents <c>)

-->
    (<j> ^empty (- <v> <c>))
}

```

This is an elaboration of the state since it does not directly apply any operator functions. Any augmentation of the state named *jug* which is itself augmented by *volume* and *contents* will have an augmentation added to hold the value of its emptiness (which is explicitly calculated by the elaboration). Whenever a change is made to the contents, the production will retract the current value of *empty* (because it is i-supported) and create an new *empty* augmentation with the correct current value of emptiness.

To solve the Water jug problem, we need to provide operators that Soar can apply to the state to change it and reach the goal. For the sake of this problem, all three

operators fill, empty and pour are indifferent to each other. Here is the production code for the fill operator:

```
#water-jug*propose*fill
#If the task is water-jug and
#there is a jug that is not full,
#then propose filling that jug.

sp {Water-jug*propose*fill
    (state <s> ^name water-jug
        ^jug.empty > 0)
-->
    (<s> ^operator <o> + =)
    (<o> ^name fill
        ^fill-jug <j>)
}

# If the problem is water jug and the fill operator is
#selected, fill a jug
sp {Water-jug*apply*fill
    (state <s> ^name water-jug
        ^operator <o>
        ^jug <j>)
    (<o> ^name fill
        ^fill-jug <j>)
    (<j> ^volume <volume>
        ^contents <contents>)
-->
    (<j> ^contents <volume>
        ^contents <contents> - ) }
```

The fill operator will be proposed for a given jug provided that the jug is not full. If the fill operator is selected, the actions of the operator removes the current contents of the jug (*^contents <contents> -*) and creates a new augmentation where the contents are equal to the volume.

A production must also be created to let Soar know when it has completed the task (otherwise it would continue filling, pouring and emptying the jugs forever!):

```
#water-jug*propose*end
#If the task is water-jug and
```

```

#the 3 litre jug contains 1 litre, stop soar.

sp {Water-jug*propose*end
    (state <s> ^name water-jug
        ^jug <j>)
    (<j> ^volume 3
        ^contents 1)
-->
    (<s> ^operator <o> !)
    (<o> ^name end)
}

# If the problem is water jug and the end operator is
#selected, the problem is solved
sp {Water-jug*apply*end
    (state <s> ^name water-jug
        ^operator.name end)
-->
    (write (crlf) | Problem soleved! |)
    (halt)
}

```

If the 3 litre jug contains 1 litre during any proposal phase, the end operator will be proposed which overrides all other operators due to its required preference. The actions of the end operator are to print 'Problem Solved' and to stop Soar.

This problem illustrates the basic sequence of operator selection and application that would be used by a game audio agent to make specific decisions and apply them.

A complete list of the productions used in this problem is provided in Appendix B

3.6.2 Eaters

Eaters is a game environment similar to pac-man, where a number of 'eaters' controlled by Soar programs move through the environment eating food and avoiding other eaters.

Figure 3.8 shows a screenshot of the game environment.

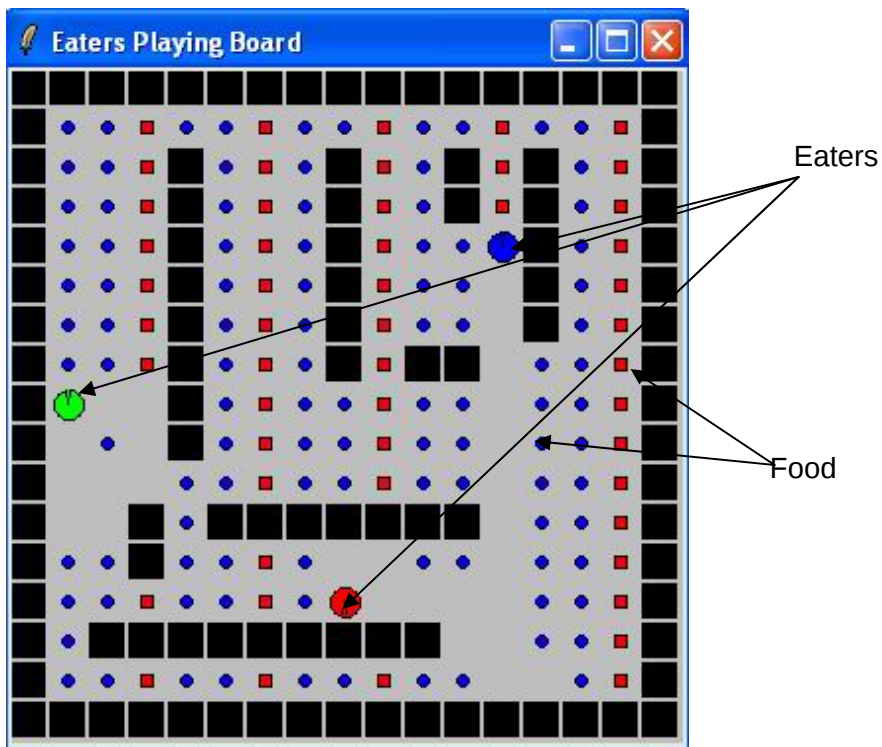


Figure 3.8 'Eaters' game environment

Each eater can sense the contents of squares up to one square past an adjacent square and can move north south east and west (moving one square at a time). The goal of the game is to eat as much food as possible without bumping into any other eaters. An agent is created for each eater with the agent's inputs set to receive information from the eater's senses and the out puts set to send information to the eater's move commands.

In order for a Soar program to interface with an external environment, productions must access the input/output link; an augmentation of the top state which is automatically created by Soar. The input/out put link could be expressed as this:

```
(state <s> ^io <io>)
(<io> ^input-link
      ^output-link)
```

In the input phase, the environment sends messages to Soar to create or update augmentations of the input-link. Similarly, in the output phase, changes that have been

made to the output link during the cycle are communicated to the environment. A simple agent would consist of an eater which moved towards food whenever it sensed it nearby and can be built using a single operator to inspect the input-link and propose a move in a direction where there is food (although there is only one operator, there may be multiple instances of operator proposals if there was food in more than one adjacent square).

The first production for this operator is a proposal which tests the input-link:

```
#If col eater exists at a position and there is adjacent
#food, propose a move towards food operator

sp {col-eater*propose*move-to-normalfood
    (state <s> ^io.input-link.my-location.<dir>.content
                                     normalfood)

-->
    (<s> ^operator <o> + =)
    (<o> ^name move-to-normalfood
        ^direction <dir>)}
```

This production tests the input link to see if the content of a square in any direction (the variable *<dir>* can match to north south east or west). If there is food in an adjacent square, the production will propose a move into that square. The preference is globally indifferent so if there is more than one possible direction, one will be selected randomly.

Once a move operator is selected, a production fires to apply it by augmenting the output link:

```
#if move-to-normalfood is selected as operator, send the
#move-north command to output link

sp {col-eater*apply*move-to-normalfood
    (state <s> ^operator (^name move-to-normalfood
                        ^direction <dir>)
        ^io.output-link <out>)

-->
    (<out> ^move.direction <dir>)}
```

The augmentation created on the output link will be communicated to the environment which will execute the move. If the command was left on the input link, the environment would try to execute the move again in the next output phase: once the environment completes an action, it sends a message to Soar to tell it that the action has been performed. A soar program has access to this information via a special augmentation on the output link:

```
#Once the move is complete, clean up the output link
sp {col-eater*apply*move-to-normalfood*remove
    (state <s> ^operator.name move-to-normalfood
        ^io.output-link <out>)
    (<out> ^move <move>)
    (<move> ^status complete)
-->
    (<out> ^move <move> -)}
```

Once the status is seen to be complete, the production will remove the augmentation from the output link.

A game audio agent must make proficient use of the input and output links in order to communicate with the game environment. An efficient design for the structures created upon the input and output link is essential in order that the agent has enough appropriate knowledge to make contextually consistent decisions and be able to implement them.

The productions for a more advanced eater which prefers 'bonusfood' to 'normalfood', avoids other eaters and will not reverse a move it just made are available in Appendix C but will not be explained in detail.

3.6.3 SML (Soar Markup Language)

SML is the application used by programs, such as a game environment, to communicate with Soar via the input output link.

SML (Soar Markup Language) provides an interface into Soar based around sending and receiving commands packaged as XML packets. The interface is designed to support connecting simulations to Soar (where input and output data structures are sent back and forth) and to support debuggers (where commands to print out specific productions or working memory elements are sent back and forth). We refer to these simulations and debuggers as “clients”.
[Anon, 2005]

A client program can include a Soar interface written in C++ or Java.

A test program was written in C++ to demonstrate simple i/o integration of Soar into an environment. A number of technical difficulties, including missing libraries, documentation errors and software version incompatibilities have prevented the completion and testing of a working Soar interface within the time limits of this investigation. The current version of the program is able to successfully create an instance of Soar and create augmentations on the input link which are updated as the program progresses. No output was received from Soar, however, and it is unclear whether this is a fault in the C++ interface or the Soar program.

When creating an SML interface to Soar within a client, a Soar kernel is first created either in the same thread as the client or in a new thread. Calling the kernel in the same thread results in faster performance but complicates the code (Anon, 2005). In the test program the kernel is called in a new thread. Once a kernel has been successfully called, an agent is created and productions loaded in. To set the initial state, structures are created on the input-link to represent all the inputs between the environment and agent. Figure 3.9 shows a flow-chart of the test program's operation.

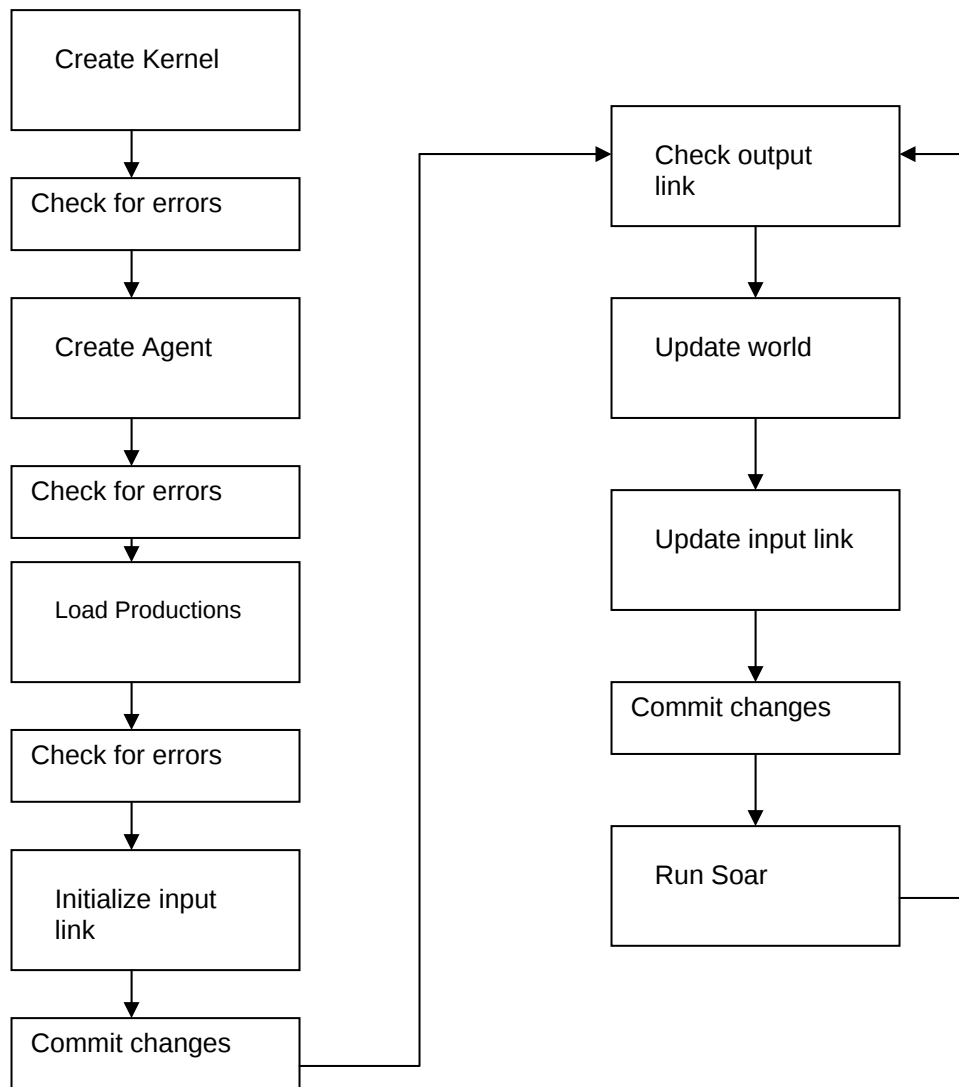


Figure 3.9 Flow diagram of SML test program

The main operation loop follows the initialization of the input link and runs continually through a sequence of checking the output from soar, updating the world, sending new input to the agent and running the agent. After any change is made to the input link, the client must *commit* them to be sent to Soar.

A breakdown of each section of code is presented below; the full unbroken code is presented in Appendix D.

In order to make use of SML, the `sml_Client.h` header file must be included.

```

//#include "sml_Connection.h"
#include "sml_Client.h"

```

Including the header file give the client access to the routines needed to create a Soar interface and call specific SML functions.

```

//Create an instance of the Soar kernel in our process
Kernel* pKernel = Kernel::CreateKernelInNewThread("SoarKernelSML");

//Check that nothing went wrong
if (pKernel->HadError())
{
    cout << pKernel->GetLastErrorDescription() << endl;
    return(0);
}

```

A Soar kernel is created in a new thread. The kernel is check for errors and if an error has occurred, the error description is printed and the program returns a value of 0, exiting the procedure.

```

//Create Soar agent
sml::Agent* pAgent = pKernel->CreateAgent("test");

//Check that nothing went wrong
if (pKernel->HadError())
{
    cout << pKernel->GetLastErrorDescription() << endl;
    return(0);
}

```

An agent named 'test' is created and checked for errors. Although the test program uses a single agent, it is possible for a kernel to support several agents.

```

//Load productions
pAgent->LoadProductions("test.soar");

//Check that nothing went wrong

```

```

if (pKernel->HadError())
{
    cout << pKernel->GetLastErrorDescription() << endl;
    return(0);
}

```

Productions are loaded into the agent from file 'test.soar' and the kernel checked for errors. A Soar production file contains text versions of all the productions to be loaded into the agent; as such it is possible to author production files in a text based editor although Visual Soar facilitates the process greatly.

```

//Get the identifier for the Soar input link
Identifier* pInputLink = pAgent->GetInputLink();

```

An identifier value for the input link is retrieved from the agent. It is crucial to know the value of the input-link's identifier so that augmentations may be created and committed to the agent.

```

//Initialize the argumentation on the input link
Identifier* pID          = pAgent->CreateIdWME(pInputLink, "action");
StringElement* pWME1     = pAgent->CreateStringWME(pID, "prod", "no");

//Send changes to working memory
pAgent->Commit();

```

The initial augmentations are created on the input link. In this case, the input link is augmented with a WME called 'action' which is itself augmented by a string-valued WME called 'prod' that has a value of 'no'. The client can call SML routines to create WME's that have either string, integer or float values.

```

//collect a value for the number of commands
    int numberCommands = pAgent->GetNumberCommands();

    //loop through all the commands from Soar and update the simulation
    for (int i = 0 ; i < numberCommands ; i++)
    {
        Identifier* pCommand = pAgent->GetCommand(i);

        std::string say = pCommand->GetParameterValue("say");

        if (say == "ouch") cout << "Ouch! \a\n" << endl;

        pCommand->AddStatusComplete();
    }

```

Commands are retrieved from Soar. The loop moves through each command in turn, retrieves the value of the augmentation 'say' on the output link and stores it in a string variable. Necessary changes are then made to the world and the output link updated with the ^status complete augmentation.

```

//Control for keyboard and send commands to Soar
    switch(KeyIn)
    {

        case '0': pAgent->Update(pWME1, "yes"); break;
        case 'q': quit = 1; break;
    }

    //Send the package to working memory
    pAgent->Commit();

    //run soar through one decision cycle
    pAgent->RunSelf(1);

```

A switch is used to manage input into Soar. If the variable 'key in' is holding the value '0', the program will update the augmentation 'prod' to have a value of 'yes'. The commands are then committed to Soar and the agent run for one cycle before looping back to check the output link for updated commands.

Although a working version of the test program was not realised during the course of the investigation, we can see from the code developed so far how an instance of Soar

could be implemented within a C++ based program. The benefit of developing the interface in C++ as opposed to Java is that a smooth development pipeline is maintained with commercial games which are mostly written in C or C++.

The production code for test.soar is included in Appendix D.

Chapter 4

Conclusion

Soar promises to be an excellent development platform for game audio AI. As games advance in technology and cultural status, demands on the quality of games and game audio is likely to increase and a solution to the challenges posed by interactive composition will be required. Using artificially intelligent agents to manage real-time composition and sound design shows good potential to provide such a solution and promises a smooth integration into the development pipeline due to the central role that AI already plays within contemporary game design.

Originally this investigation began as an interactive composition which would place the audience in a surround sound, sonic environment which they could explore and, through their actions, influence. The composition was to have a varied cycle over the course of twenty four hours and would include a number of events which would occur at different times of day and would change depending on the audience's actions. A method of controlling these events which would allow complicated relationships between the events and audience to be expressed. At this point it became obvious that the composition was becoming more and more similar to a computer game with the emphasis on interactive development of events and audio rather than following a script. Research into possible AI techniques suggested that Soar architecture was the most promising platform for development; having more than enough power to manage events within the composition in addition to a history of involvement with games and current research being focussed on plot-based AI.

It soon became apparent that Soar was capable of handling far more advanced audio techniques than would be implemented within the composition. Research into computer game audio indicated that there were significant challenges for interactive composition that could possibly be met by an AI audio agent. Additionally, the growing cultural and technological significance of computer games was recognised and in the light of these factors, a decision was made to change the objective of the research to an investigation of the possible application of AI within game audio.

An analysis of current theory, practise and technology in game audio revealed that while positive steps had been made towards improving audio in games, such as the DirectMusic Producer software, game audio was still not providing the same sense of emotional immersion found in other media such as films. The major challenge of providing high quality audio for games was found to be the implementation of linear techniques found in traditional music; particularly film music and sound which serves as a role-model for game audio. A comparison between films and games highlighted the fact that there are many parallels between the two genres, but that audio in games is considered subservient to the picture; a concept common in early film sound theory but now considered obsolete. The most significant techniques used in composition and sound design for films, but rarely in games, were found to be the creative use of asynchronous sound, transitions and structure; particularly in relation to the development of the plot. It was proposed that an AI audio agent could provide a solution to implementing these techniques in an interactive environment given that one of the primary aims of AI research was to develop agents capable of modelling human behaviour.

A number of AI techniques, including neural networks and first order logic, were considered with regards to their potential as a development platforms for game audio agents. It was decided that finite-state machines, production based systems and

semantic networks were the most appropriate for initial research due to their simplicity and strong relationships with current game design practise. The Soar architecture was identified as supporting all three methods of AI design. An introduction into the design and functionality of Soar was given and highlighted the ability of Soar to effectively model human behaviours such as problem solving and decision making using sub-goals. Previous research into using Soar agents within games was found to have provided a rich source of techniques, which could be directly applied to audio agents. Planning and learning techniques implemented in Soar Quakebot could be used to anticipate changes in the game state and provide more musical transitions, particularly when using an operator hierarchy similar to Quakebot; allowing the agent to be reactive but retain a sense of context. Although the interface specification from Quakebot is incompatible with that of an audio agent, the lessons learnt about the importance of a well designed interface are still significant. Current research into application of Soar to enhance plot-based game experience with an Interactive Drama Architecture provides a possible blueprint for audio agents. These agents would be able to use sound design and music throughout a game to both follow the development of the plot and actively promote the plot by influencing the player's decisions and emotions.

Future investigation to continue this research should primarily focus on developing a fully functional interface with Soar and a number of individual Soar programs to manage the different aspects of creative real-time audio, including the use of asynchronous sound, plot related structure and transitions. Different methods of realising these goals should be investigated including (but not limited to) using anticipation and behavioural modelling to predict the actions of the player, using state-based plot to provide an abstract structure and using hierarchical operator systems with sub-goaling.

An alternative approach would be to investigate the possibilities of other AI architectures more thoroughly. Neural networks in particular are very promising for behavioural simulation due to their powerful learning capacity.

References

- Ames, C. (1990) *The Age of Intelligent Machines*. Internet WWW page, at URL <<http://www.kurzweilai.net/articles/art0305.html?printable=1>> (version current at 8 August 2005)
- Anon (2005) *SML Quick Start Guide*. Internet WWW resource, at URL <http://sitemaker.umich.edu/soar/soar_software_downloads> as component of Soar Suite 6.1
- Cawsey, A. (1998) *The Essence of Artificial Intelligence*. Hertfordshire, Prentice Hall
- Chamandard, A. (2004) *AI Game Development Synthetic Creatures with Learning and Reactive Behaviours*. USA, New Riders Publishing
- Chion, M. (1994) *Acoustimatic Sound*. Internet WWW page, at URL <<http://hem.passagen.se/filmljud/filmsound.htm>>
- Cope, D. (2000) *The Algorithmic Composer*. Madison, A-R Editions, Inc.
- Copeland, J. (1993) *Artificial Intelligence a Philosophical Introduction*. Massachusetts, Blackwell Publishers.
- Deloura, M. (ed.) (2001) *Game programming Gems 2*. Massachusetts, Charles River Media, INC.
- de Man, J. (2005) *Interview with Joris de Man, Sound Designer for Guerrilla, the Developer of Killzone*. Internet WWW page, at URL <http://dolby.com/consumer/games/interview_04.html>
- Dodge, C & Jerse, T (1997) *Computer Music*. New York: Schirmer Books
- Emered-Johnson, J (2005) Cited by Waugh, E. (2005b) *GDC 2005 Report: What Makes Music for Games "Music for Games"?* Internet WWW page, at URL <<http://www.gamasutra.com/gdc2005/features/20050318/postcard-waugh.htm>>
- www.enterthematrixgame.com. Homepage for Enter the Matrix. Internet WWW page, at URL: <<http://www.enterthematrixgame.com>>(version current at 24 July 2005)
- Goldwasser, D. (2005) *Live music for Games*. Internet WWW page, at URL <<http://www.soundtrack.net/features/article/?id=155>>
- Greeves, D. (2005) *Grand Theft Audio*. Sound on Sound, February 2005. Cambridge, Sound on Sound Limited
- Griffin, S. (1998) *Musical Techniques for Interactivity*. Internet WWW page, at URL <http://www.gamasutra.com/features/sound_and_music/19980501/interactivity_techniques_06.htm>

Jordan, J. (2005) *Develop*. Hertford, Intent Media

Kane, B. (2004) *Why We Play Games: The Four Keys to Player Experience*. WWW page, at URL <http://www.gamasutra.com/gdc2004/features/20040326/postcard-kane_04.shtml>

Laird, J. (2002) *Research in Human-level AI using Computer Games* Internet WWW page, at URL: <http://ai.eecs.umich.edu/people/laird/papers/CACM01.pdf>

Laird, J. (2005) *Soar 8 Tutorial*. Internet WWW resource, at URL <http://sitemaker.umich.edu/soar/soar_software_downloads> as component of Soar Suite 6.1

Laird, J. & Congdon, C. (2005) *Soar User's Manual Version 8.6*. Internet WWW resource, at URL <http://sitemaker.umich.edu/soar/soar_software_downloads> as component of Soar Suite 6.1

Laird, J. & Jones, R. (1998) *Building Advanced Autonomous AI Systems for Large Scale Real Time Simulations*, Proceedings of the Computer Game Developers Conference, May 1998, Long Beach, CA

Laird, J. & Lent, M. (1999) *Developing an Artificial Intelligence Engine*. Proceedings of the Game Developers Conference, March 16-18, 1999, San Jose, CA, pp. 577-588

Langston, P. (2005) *Lucasfilm Games*. Internet WWW page, at URL <<http://www.langston.com/LFGames/>>

Leyton, C. (2004) *GTA: San Andreas 150 Hours Gameplay!!!* Internet WWW page, at URL<http://www.totalvideogames.com/news/GTA: San Andreas 150 Hours Gameplay!!!_6063_2972_0.htm>

Magerko, B. & Laird, J. (2003) *Building an Interactive Drama Architecture*, 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment, Darmstadt, Germany, March 24 – 26

Marks, A. (2003) *Game Audio: A GDC 2003 Wrap-up*. Internet WWW page, at URL <http://www.gamasutra.com/features/20030313/marks_01.shtml>

McDonald, G. (2004) *A History of Video Game Music*. Internet WWW resource, at URL <http://www.gamespot.com/features/6092391/index.html>

Menshikov, A. (2003) *Modern Audio Technologies in Games*. Internet WWW page, at URL <<http://www.digit-life.com/articles2/sound-technology/>>

Morton, S. (2005a) *Enhancing the Impact of Music in Drama Orientated Games*. Internet WWW page, at URL <http://www.gamasutra.com/features/20050124/morton_01.shtml>

Morton, S. (2005b) *Dramatic Music in Computer Games*. Personal correspondence, Appendix A

Moser, M. (1994) *Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing*. Internet WWW resource, at URL <<ftp://ftp.cs.colorado.edu/users/mozer/papers/music.pdf>>

Nuxoll, A. & Laird, J. (2003) *Soar Design Dogma*. Internet WWW resource, at URL <<http://www.eecs.umich.edu/~soar/sitemaker/docs/misc/dogma.pdf>>

Patterson, S. (2001) *Game programming Gems 2*. Massachusetts, Charles River Media, INC.

Rabin, S. (2002) *AI Game Programming Wisdom*. Massachusetts, Charles River Media, INC

Rosen, C. (1988) *Sonata Forms*. Ontario, Norton & Company, Inc

Rockville, M. (2004) *Oblivion Overview*. Internet WWW page, at URL <http://www.elderscrolls.com/games/oblivion_overview.htm>

Rogers, J (1997) *Object Oriented Neural Networks in C++*. San Diego, Academic Press, Inc

Rona, J. (2000) *the Reel World Scoring for Pictures*. San Francisco, Miller Freeman.
Rossing, T. et al. (2002) *The Science of Sound*. San Fransico, Addison Wesley

Russel, S. & Norvig, P. (2003) *Artificial Intelligence A Modern Approach*. New jersey, Pearson Education, Inc

Schneider, P. (2004) *Dear Friends: Music From Final Fantasy*. Internet WWW page, at URL <<http://music.ign.com/articles/513/513292p1.html>>

Song, J. (2005) *Improving the Combat Impact of Action Games*. Internet WWW page, at URL <http://www.gamasutra.com/features/20050428/sang_02.shtml>

Sonnenschien, D. (2001) *Sound Design The Expressive Power of Music, Voice and Sound Effects in the Cinema*. Studio City, Michael Wiese Productions
Sony, (2003) see ch1 – re conference

Stevens, R. (2001) *Interactive audio for Computer Games*. Unpubl. MSc thesis. York University.

Tallarico, T. (2005) *Tommy Tallarico on His Career, The State of Game Music and More*. Internet WWW page at URL:
<http://biz.gamedaily.com/features.asp?article_id=9315&filter=interview>

Treglia, D. (ed.) (2002) *Game Programming Gems 3*. Massachusetts, Charles River Media, INC.

Tipler, P. (1999) *Physics for Scientists and Engineers*. New York, W.H. Freeman and Company

Watchowski Brothers. (2005) *The Matrix Trilogy*. Internet WWW page, at URL: <<http://whatisthematrix.warnerbros.com>>(version current at 23 July 2005)

Waugh, E. (2005a) *GDC 2005 Report: Audio Production for Halo 2*. Internet WWW page, at URL <<http://www.gamasutra.com/gdc2005/features/20050310/postcard-waugh.htm>>

Waugh, E. (2005b) *GDC 2005 Report: What Makes Music for Games "Music for Games"?* Internet WWW page, at URL <<http://www.gamasutra.com/gdc2005/features/20050318/postcard-waugh.htm>>

Weis, E. & Belton, J. (eds) (1985) *Film Sound Theory and Practice*. New York, Columbia University Press

White, P. (2003) *Creative Recording part one effects and processors*. London, Sanctuary Publishing Ltd.

Whitmore, G. (2003) *Design With Music in Mind: a Guide to Adaptive Audio for Game Designers*. Internet WWW page, at URL <http://www.gamasutra.com/resource_guide/20030528/whitmore_02.shtml>

Wilde, M. (2004) *Introduction to Interactive Game Audio*. Internet WWW page, at URL: <<http://www.digitalproducer.com/articles/viewarticle.jsp?id=26180>>

Appendix A

Correspondence with Scott Morton

scott@scottbmorton.com

Dramatic music in Computer Games

Hi Col,

Thanks for writing; glad you found the article useful. I'm not entirely sure what specifics you're meaning when you say "using AI for music and sfx." I'll take a guess and say that you are talking about having the game reconstruct tracks or pull from a database of audio based on individual events or states in the game.

In my opinion, it really depends on the game and how the gameplay works that determines how music and sound should be implemented and function. If your gameplay is linear and contains dramatic choke points, you'd probably be better off just coming up with good material and spacing it properly with simple triggers throughout levels instead of having it dynamically reconstructed. You could add a little interactivity by using multiple streams of audio, thereby giving the music the ability to shift intensity using layer techniques. This could help in situations where the music needs to adapt to player location, etc.

In games that are more open ended (Elder Scrolls comes to mind) you could benefit from an interactive music system that builds compositions using musical components (cells, streams, what have you) based on where you are/what you are doing. The advantage of using a database approach is that you could greatly randomize different combinations of cells/streams and dynamically create new music for different situations. Those situations could simply be defined as variable states in the game engine or however programmers/designers decided to do it, and these states could call preset combinations that you yourself as the audio designer put together in the database.

The challenge of course, when designing interactive music, is creating components that fit together in multiple combinations. It's a lot harder than it seems and requires a much more involved approach than simple linear scoring. It's just as important to create material that works together as it is the game engine's ability to reassemble it properly. So there's definitely that need for an audio programmer and dedicated design attention to the audio part of the game.

I'm a busy guy but I'll see if I can take a little time to look at your paper once you're finished and give you my thoughts. Hope this email helps a bit. =).

Take care, Scott

Appendix B

Production code for 'The Water Jug' problem

The following code is a list of all the productions used in the Water Jug problem (based on the outline given in the Soar 8 Tutorial (Laird, 2005)).

#If no task is selected, propose the water jug task

```
sp {propose*initialize-Water-jug
  (state <s> ^superstate nil
    -^name)
-->
  (<s> ^operator <o> +)
  (<o> ^name initialize-Water-jug)
}
```

#if the water jug task is selected:

initialize the water-jug problem

```
sp {apply*initialize-Water-jug
  (state <s> ^operator.name initialize-Water-jug)
-->
  (<s> ^name water-jug
    ^jug <j1>
    ^jug <j2>)
  (<j1> ^volume 5
    ^contents 0)
  (<j2> ^volume 3
    ^contents 0)
}
```

If the state is named water-jug and a
jug can hold volume v and currently has
contents c, then add that it has $v - c$
available (empty) space.

```
sp {Water-jug*elaborate*empty
  (state <s> ^name water-jug
    ^jug <j>)
```



```

    (<j> ^volume <v>
      ^contents <c>)

-->
    (<j> ^empty (- <v> <c>))
}

#water-jug*propose*empty
#If the task is water-jug and
#there is a jug that is not empty,
#then propose emptying that jug.

sp {Water-jug*propose*empty
    (state <s> ^name water-jug
      ^jug.contents > 0)
-->
    (<s> ^operator <o> + =)
    (<o> ^name empty
      ^empty-jug <j>)
}

# If the problem is water jug and the empty operator is selected for a jug, empty the jug
sp {Water-jug*apply*fill
    (state <s> ^name water-jug
      ^operator <o>
      ^jug <j>)
    (<o> ^name empty
      ^empty-jug <j>)
    (<j> ^volume <volume>
      ^contents <contents>)
-->
    (<j> ^contents 0
      ^contents <contents> -)}

#water-jug*propose*fill
#If the task is water-jug and
#there is a jug that is not full,
#then propose filling that jug.

sp {Water-jug*propose*fill
    (state <s> ^name water-jug
      ^jug.empty > 0)
-->
    (<s> ^operator <o> + =)
    (<o> ^name fill
      ^fill-jug <j>)
}

```

If the problem is water jug and the fill operator is selected, fill a jug

```
sp {Water-jug*apply*fill
  (state <s> ^name water-jug
    ^operator <o>
    ^jug <j>)
  (<o> ^name fill
    ^fill-jug <j>)
  (<j> ^volume <volume>
    ^contents <contents>)
-->
  (<j> ^contents <volume>
    ^contents <contents> -)}
```

#water-jug*propose*pour
#If the task is water-jug and there is a
#jug that is not full and the other
#jug is not empty,then propose
#pouring water from the second jug
#into the first jug.

```
sp {Water-jug*propose*pour
  (state <s> ^name water-jug
    ^jug <j>
    ^jug {<i> <> <j>})
  (<i> ^empty > 0)
  (<j> ^contents > 0)
-->
  (<s> ^operator <o> + =)
  (<o> ^name pour
    ^fill-jug <i>
    ^empty-jug <j>)
}
```

#if the pour operator is selected and the pouring jug can empty
#itself into the other:

```
sp {water-jug*apply*pour*will-empty-empty-jug
  (state <s> ^name water-jug
    ^operator <o>)
  (<o> ^name pour
    ^empty-jug <i>
    ^fill-jug <j>)
  (<j> ^volume <jvol>
    ^contents <jcon>
    ^empty <jempty>)
  (<i> ^volume <ivol>
    ^contents { <icon> <= <jempty> })
-->
  (<i> ^contents 0
    ^contents <icon> -)
```

```

    (<j> ^contents (+ <jcon> <icon>)
      ^contents <jcon> -)}

#if the pouring jug cannot empty itself

sp {water-jug*apply*pour*will-not-empty-empty-jug
  (state <s> ^name water-jug
    ^operator <o>)
  (<o> ^name pour
    ^empty-jug <i>
    ^fill-jug <j>))
  (<i> ^volume <ivol>
    ^contents { <icon> > <jempty> })
  (<j> ^volume <jvol>
    ^contents <jcon>
    ^empty <jempty>))
-->
  (<i> ^contents (- <icon> <jempty>)
    ^contents <icon> -)
  (<j> ^contents <jvol>
    ^contents <jcon> -)}

#water-jug*propose*end
#If the task is water-jug and
#the 3 litre jug contains 1 litre, stop soar..

sp {Water-jug*propose*end
  (state <s> ^name water-jug
    ^jug <j>)
  (<j> ^volume 3
    ^contents 1)
-->
  (<s> ^operator <o> !)
  (<o> ^name end)
}

# If the problem is water jug and the end operator is selected, the problem is solved
sp {Water-jug*apply*end
  (state <s> ^name water-jug
    ^operator.name end)
-->
  (write (crLf) | Problem soleved! |)
  (halt)
}

```

Appendix C

Advanced 'cool-eater' productions

The productions for this eater specify a single, generic move operator which identifies the direction of movement and the content of the destination. Separate productions create preferences for operators depending on the destination content. The advantage of abstracting the preferences in this way is that the behaviour of the eater can be easily expanded or refined by adding or changing only the preferences and not the entire move operator.

#if bonusfood is adjacent propose to move towards it, and is more preferable than normal food

```
sp {propose*move*2a
  (state <s> ^io.input-link.my-location.<dir>.content
    { <content> <> wall })
-->
  (<s> ^operator <o> + =)
  (<o> ^name move
    ^direction <dir>
    ^content <content>}}
```

#if move-to-bonusfood is selected as operator, send the move-to-bonus command to output link

```
sp {apply*move
  (state <s> ^io.output-link <out>
    ^operator <o>)
  (<o> ^name move
    ^direction <dir>)
-->
  (<out> ^move.direction <dir>}}
```

```
# Apply*move*remove-move:
# If the move operator is selected,
# and there is a completed move command on the output link,
# then remove that command.
```

```
sp {apply*move*remove-move
  (state <s> ^io.output-link <out>
    ^operator.name move)
  (<out> ^move <direction>)
  (<direction> ^status complete)
-->
```

```

(<out> ^move <direction> -)}

# Apply*move*create*last-direction
# If the move operator for a direction is selected,
# create an augmentation called last-direction with that direction.
sp {apply*move*create*last-direction
  (state <s> ^operator <o>)
  (<o> ^name move
    ^direction <direction>)
-->
  (<s> ^last-direction <direction>)}

# Apply*move*remove*last-direction
# If the move operator for a direction is selected,
# and the last-direction is not equal to that direction,
# then remove the last-direction.
sp {apply*move*remove*last-direction
  (state <s> ^operator <o>
    ^last-direction <direction>)
  (<o> ^direction <> <direction>
    ^name move)
-->
  (<s> ^last-direction <direction> -)}

# Bonusfood is a better choice than normalfood, an empty square, or a square with
#another eater in

sp {select*move*bonusfood-better-than-normalfood-empty-eater
  (state <s> ^operator <o1> +
    ^operator <o2> +)
  (<o1> ^name move
    ^content bonusfood)
  (<o2> ^name move
    ^content << normalfood empty eater >>)
-->
  (<s> ^operator <o1> > <o2>)}

#Normalfood is a better choice than either an empty square or a square with another
#eater in
sp {select*move*normalfood-better-than-empty-eater
  (state <s> ^operator <o1> +
    ^operator <o2> +)
  (<o1> ^name move
    ^content normalfood)
  (<o2> ^name move
    ^content << empty eater >>)
-->

```

```
(<s> ^operator <o1> > <o2>))}
```

```
#An empty square is better than a square with an eater in  
sp {select*move*empty-better-than-eater
```

```
  (state <s> ^operator <o1> +  
    ^operator <o2> +)
```

```
  (<o1> ^name move  
    ^content empty)
```

```
  (<o2> ^name move  
    ^content eater)
```

```
-->
```

```
  (<s> ^operator <o1> > <o2>))}
```

```
# Select*move*reject*backward
```

```
# If there is a proposed operator to move in the direction
```

```
# opposite the last move,
```

```
# reject that operator.
```

```
sp {select*move*reject*backward
```

```
  (state <s> ^operator <o> +  
    ^directions <d>  
    ^last-direction <dir>)
```

```
  (<d> ^value <dir>  
    ^opposite <o-dir>)
```

```
  (<o> ^name move  
    ^direction <o-dir>)
```

```
-->
```

```
  (<s> ^operator <o> -))}
```

Appendix D

Production code used in SML test program

The production code for test.soar is a simple routine to inspect the input link and wait for the augmentation ^action.prod yes. When the condition matches, an operator is selected to place the augmentation ^voice.say.ouch on the output link.

```
#If I exist and the simulation prods me propose that I say ouch
```

```
sp {test*propose*ouch
  (state <s> ^name test
    ^io.input-link.action.prod yes)
-->
  (<s> ^operator <o> +)
  (<o> ^name ouch)}
```

```
# If I exist and the ouch operator is selected, tell the simulation 'ouch'
```

```
sp {test*apply*ouch
  (state <s> ^name test
    ^operator.name ouch)
-->
  (<s> ^io.output-link.voice.say ouch)}
```

```
#If I have said ouch, remove ouch from the outputlink
```

```
sp {apply*move*remove-move
  (state <s> ^io.output-link <out>
    ^operator.name ouch)
  (<out> ^voice <com>)
  (<com> ^status complete)
-->
  (<out> ^voice <com> -)}
```